Министерство науки и высшего образования Российской Федерации

Томский государственный университет
систем управления и радиоэлектроники

С. Ю. Золотов

# ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННЫХ СИСТЕМ

Учебное пособие
для студентов направления подготовки
09.03.03 «Прикладная информатика» (уровень бакалавриата),
профиль «Прикладная информатика в экономике»

Томск
2023

**УДК** 004.942
**ББК** 30.2-5-05
З-80

**Рецензент**:
**Пермякова Н.В.**, доцент кафедры автоматизации обработки информации ТУСУР, канд. техн. наук

**Золотов, Сергей Юрьевич**
З-80 Проектирование информационных систем: учебное пособие для студентов направления подготовки 09.03.03 «Прикладная информатика» (уровень бакалавриата), профиль «Прикладная информатика в экономике» / С. Ю. Золотов. – Томск : Томск. гос. ун-т систем упр. и радиоэлектроники, 2023. – 61 с.

В учебном пособии раскрываются теоретические основы проектирования информационных систем, особое внимание уделяется описанию структурного и объектно-ориентированного подходов к проектированию информационных систем. Данное учебное пособие предназначено для студентов всех форм обучения направления 09.03.03 «Прикладная информатика» (уровень бакалавриата).

Одобрено на заседании каф. АСУ, протокол № 11 от 23 ноября 2023 г.

УДК 004.942
ББК 30.2-5-05

**Оглавление**

Введение ……... 4
1 Общие сведения об информационных системах ……... 5
1.1 Понятие информации …… 5
1.2 Понятие информационных систем …….. 5
2 Методологические основы проектирования информационных систем ……. 8
2.1 Технология проектирования информационных систем …… 8
2.2 Принципы проектирования сложных систем ……. 9
2.3 Классификация типовых проектных процедур …….. 12
3 Структурный подход к проектированию информационных систем ……... 17
3.1 Сущность структурного подхода ……. 17
3.2 Методология функционального моделирования IDEF0 …… 17
4 Объектно-ориентированный подход к проектированию информационных систем 26
4.1 Общие сведения об объектно-ориентированном проектировании информационных систем ……... 26
4.2 Объектно-ориентированные концепции ……. 28
4.3 Моделирование классов ……... 30
4.4 Моделирование состояний ……... 42
4.5 Моделирование взаимодействий ……. 50
Заключение …….. 57
Глоссарий ……. 58
Список литературы ……. 61

## Введение

Без информационных систем уже немыслима жизнь современного человека. Информационные технологии прочно проникли во все сферы деятельности людей. Для разработки и поддержания в работоспособном состоянии информационных систем требуются знания об их проектировании.

Знания в области проектирования информационных систем включают в себя знания системного анализа прикладной области, формализации решения прикладных задач и процессов информационных систем (ИС), разработку требований к созданию и развитию ИС и ее компонентов, разработку проектов автоматизации и информатизации прикладных процессов и создание ИС в прикладных областях, а также управление проектами информатизации предприятий и организаций.

Данное учебное пособие предназначено для получения основных знаний по проектированию информационных систем. Для успешного усвоения дисциплины «Проектирование информационных систем» студент должен обладать знаниями системного анализа, общими сведениями об информационных технологиях, навыками программирования.

# 1 Общие сведения об информационных системах

## 1.1 Понятие информации

Прежде чем говорить об информации, необходимо дать определение энтропии [1].

***Энтропия*** – это мера неопределенности какого-либо опыта, который может иметь разные исходы.

Основные свойства энтропии:

1) Энтропия есть величина неотрицательна.

2) Энтропия есть величина ограничена.

3) Энтропия равна нулю только тогда, когда одна из вероятностей равна единице, а другие вероятности равны нулю.

4) Энтропия достигает своего максимального значения тогда, когда все события равновероятны.

5) Энтропия нескольких независимых источников равна сумме энтропии этих источников.

Принято говорить, что один объект отражает другой или содержит информацию о другом объекте, если состояние одного объекта находится в соответствии с состоянием другого объекта.

***Информация*** есть свойство материи, состоящее в том, что в результате взаимодействия объектов между их состояниями устанавливается определенное соответствие. Чем сильнее выражено это соответствие, тем точнее состояние одного объекта отражает состояние другого объекта, т.е. тем больше информации один объект содержит о другом.

В настоящее время информация рассматривается как фундаментальное свойство материи.

***Сигнал*** – материальный носитель информации, т.е. средство переноса информации в пространстве и во времени. Один и тот же объект может выступать в роли разных сигналов. В качестве сигналов используются не сами объекты, а их состояния.

***Сообщение*** – совокупность знаков или периодических сигналов, содержащих информацию. Одно и тоже сообщение ведет к разной интерпретации, т.е. к разной информации.

***Количество информации*** – это мера уменьшения энтропии объекта после совершения некоторого события.

Единица информации – один бит (bit – BInery digiT).

***Один бит*** – это количество информации, получаемое при осуществлении одного из двух равновероятных событий.

Основные производные единицы количества информации: 1 байт = 8 бит; 1Кбайт = 1024 байт; 1 Мбайт = 1024 Кбайт; 1 Гбайт = 1024 Мбайт; 1 Тбайт = 1024 Гбайт.

## 1.2 Понятие информационных систем

Существует множество определений термина система [1, 2].

Первое определение системы: система есть средство достижения цели.

Другое определение: система – это совокупность взаимосвязанных элементов, обособленное от среды и взаимодействующая с ней как единое целое.

Можно дать и обобщенное определение:

***Система*** – это конечное множество функциональных элементов и отношений между ними, выделяемое из среды в соответствии с определенной целью в рамках определенного временного интервала.

***Элемент системы*** – некоторый объект, обладающий рядом важных свойств и реализующий в системе определенный закон функционирования, причем, внутренняя структура данного объекта не рассматривается.

***Подсистема*** – это относительно независимая часть системы, которая обладает всеми свойствами системы и, в частности, имеет свою подцель, на достижение которой эта подсистема и ориентирована.

Если же части системы не обладают свойствами системы, а представляют собой просто совокупности однородных элементов, то такие части принято называть ***компонентами***.

Под ***свойством*** понимают сторону объекта, обуславливающую его отличие от других объектов или сходство с ними и управляющуюся при взаимодействии с другими объектами.

Под ***управлением*** в самом общем виде будем понимать процесс формирования целенаправленного поведения системы посредством информационных воздействий, вырабатываемых человеком или устройством [2].

Система с управлением включает три подсистемы: управляющую систему (УС), объект управления (ОУ) и систему связи (СС). Управляющая система и система связи образуют систему управления (СУ). Основным элементом организационно-технической системы управления является лицо, принимающее решение (ЛПР).

Основными группами функций СУ являются [2]:

1) **Функции принятия решений.** Эти функции выражаются в создании новой информации в ходе анализа, планирования и оперативного управления. Это связано с преобразованием информации о состоянии ОУ и внешней среды в управляющую информацию при решении задач и выполнении аналитических расчетов, проводимых ЛПР при порождении и выборе альтернатив. Эта группа функций является главной, поскольку обеспечивает выработку информационных воздействий по удержанию в существующем положении или при переводе системы в новое состояние.

2) **Функции обработки информации.** Они охватывают учет, контроль, хранение, поиск, отображение, копирование информации. Эта группа функций по обработке информации не изменяет смысл самой информации (рутинные функции).

3) **Функции обмена информацией.** Эта группа функций связана с доведением выработанных воздействий ЛПР до ОУ.

Совокупность средств информационной техники и людей, объединенных для достижения определенных целей, в том числе и для управления, образуют информационную систему.

***Информационная система*** (ИС) – это организационно-техническая система, использующая информационные технологии в целях обучения, информационно-аналитического обеспечения человеческой деятельности и процессов управления [2].

## Контрольные вопросы к 1 главе

1. Что такое энтропия?
2. Что представляет понятие информации?
3. Чем отличается сигнал от сообщения?

4. Как рассчитать количество информации?
5. Чему равна минимальная единица количества информации?
6. Что такое «система»?
7. Чем подсистема отличается от компоненты системы?
8. В чем состоит особенность элемента системы?
9. Что понимается под понятием «управление системой»?
10. Какие подсистемы входят в систему с управлением?
11. Какие бывают группы функций системы с управлением?
12. Что такое «информационная система»?

# 2 Методологические основы проектирования информационных систем

## 2.1 Технология проектирования информационных систем

Современные информационные технологии предоставляют широкий выбор способов реализации ИС, выбор которых осуществляется на основе требований со стороны предполагаемых пользователей, которые, как правило, изменяются в процессе разработки. Для теории принятия решений процесс проектирования ИС – это процесс принятия проектно-конструкторских решений, направленных на получение проекта системы, удовлетворяющего требованиям заказчика [3].

Под ***проектом*** ИС будет понимать проектно-конструкторскую и технологическую документацию, в которой представлено описание проектных решений по созданию и эксплуатации ИС в конкретной программно-технической среде.

Под ***проектированием*** ИС понимается процесс преобразования входной информации об объекте проектирования, о методах проектирования и об опыте проектирования объектов аналогичного назначения в соответствии со стандартами в проект ИС.

С этой точки зрения проектирование ИС сводится к последовательной формализации проектных решений на различных стадиях жизненного цикла ИС.

Объектами проектирования ИС являются отдельные элементы или их компоненты функциональных и обеспечивающих частей. Так, функциональными элементами в соответствии с традиционной декомпозицией выступают задачи, комплексы задач и функции управления. В составе обеспечивающей части ИС объектами проектирования служат элементы и их компоненты информационного, программного и технического обеспечения системы.

В качестве субъекта проектирования ИС выступают коллективы специалистов, которые осуществляют проектную деятельность, как правило, в составе специализированной проектной организации, и организация-заказчик, для которой необходимо разработать ИС. При большом объеме и жестких сроках выполнения проектных работ в разработке системы может принимать участие несколько проектных коллективов. В этом случае выделяется головная организация, которая координирует деятельность всех организаций-соисполнителей.

Осуществление проектирования ИС предполагает использование проектировщиками определенной технологии проектирования, соответствующей масштабу и особенностям разрабатываемого проекта.

***Технология проектирования*** – это совокупность методологии и средств проектирования ИС, а также методов и средств организации проектирования.

В основе технологии проектирования лежит технологический процесс, который определяет действия, их последовательность, состав исполнителей, средства и ресурсы, требуемые для выполнения этих действий. Так, технологический процесс проектирования ИС в целом делится на совокупность последовательно-параллельных, связанных и соподчиненных цепочек действий. Действия, которые выполняются при проектировании ИС, могут быть определены как неделимые технологические операции или как подпроцессы технологических операций. Все действия могут быть собственно проектировочными, которые формируют или модифицируют результаты проектирования, и оценочными действиями, которые вырабатывают по установленным критериям оценки результатов проектирования.

Таким образом, технология проектирования задается регламентированной последовательностью технологических операций, выполняемых в процессе создания проекта на основе того или иного метода, в результате чего стало бы ясно, не только ЧТО должно быть сделано для создания проекта, но и КАК, КОМУ и в КАКОЙ ПОСЛЕДОВАТЕЛЬНОСТИ это должно быть сделано.

К основным требованиям, предъявляемым к выбираемой технологии проектирования, относятся следующие:

– созданный с помощью этой технологии проект должен отвечать требованиям заказчика;

– выбранная технология должна максимально отражать все этапы цикла жизни проекта;

– выбираемая технология должна обеспечивать минимальные трудовые и стоимостные затраты на проектирование и сопровождение проекта;

– технология должна быть основой связи между проектированием и сопровождением проекта;

– технология должна способствовать росту производительности труда проектировщика;

– технология должна обеспечивать надежность процесса проектирования и эксплуатации проекта;

– технология должна способствовать простому ведению проектной документации.

Основу технологии проектирования ИС составляет методология, которая определяет сущность, основные отличительные технологические особенности. Методология проектирования предполагает наличие некоторой концепции, принципов проектирования, реализуемых наборов методов проектирования, которые, в свою очередь, должны поддерживаться некоторыми средствами проектирования.

Организация проектирования предполагает определение методов взаимодействия проектировщиков между собой и с заказчиком в процессе создания проекта ИС, которые могут также поддерживаться набором специфических средств.

Для конкретных видов технологий проектирования свойственно применение определенных средств разработки ИС, которые поддерживают выполнение как отдельных проектных работ, этапов, так и их совокупностей. Поэтому перед разработчиками ИС, как правило, стоит задача выбора средств проектирования, которые по своим характеристикам в наибольшей степени соответствуют требованиям конкретного предприятия.

Средства проектирования должны быть:

– в своем классе инвариантными к объекту проектирования;

– охватывать в совокупности все этапы жизненного цикла ИС;

– технически, программно и информационно совместимыми;

– простыми в освоении и применении;

– экономически целесообразными.

### 2.2 Принципы проектирования сложных систем

При проектировании сложных систем используются следующие принципы:

– декомпозиция и иерархичность построения описаний объектов проектирования;

– многоэтапность и итерационность процесса проектирования;

– типизация и унификация проектных решений.

Описания технических систем должны быть по сложности согласованы: 1) с возможностями восприятия человеком; 2) с возможностями оперирования описаниями в процессе их преобразования с помощью имеющихся средств проектирования.

Выполнить это требование в рамках единого описания удается лишь для простых систем. Как правило, требуется структурирование описаний и соответствующее разбиение представлений о системе на иерархические уровни и аспекты. Это позволяет распределить работы по проектированию сложных систем между подразделениями проектировщиков, что способствует повышению эффективности и производительности труда.

Разделение описаний по степени детализации отображаемых свойств и характеристик систем лежит в основе блочно-иерархического подхода к проектированию и приводит к появлению иерархических уровней (уровней абстрагирования) в представлениях о системе (рис. 2.1).

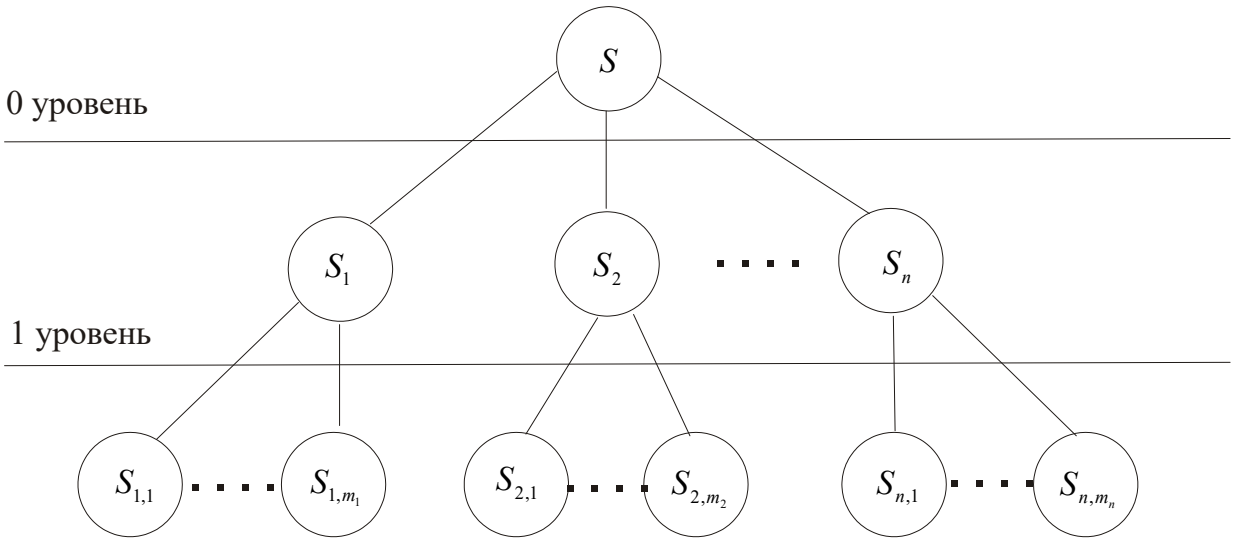


Рисунок 2.1 – Блочно-иерархическое разделение сложной системы

На уровне 0 (верхнем уровне) сложная система рассматривается как состоящая из $n$ взаимно связанных и взаимодействующих элементов (как правило, подсистем) на уровне 1.

Каждый из элементов в описании уровня 1 представляет собой также довольно сложный объект, который, в свою очередь, рассматривается как описание подсистемы на уровне 2. Элементами подсистемы являются объекты. Как правило, выделение элементов происходит по функциональному признаку.

Подобное разбиение продолжается вплоть до получения на некотором уровне элементов, описания которых дальнейшему делению не подлежат, то есть до элементов, описание которых уже известно. Такие элементы по отношению к системе называются ***базовыми*** элементами.

***Принцип иерархичности*** означает структурирование представлений о системе проектирования по степени детальности описаний (уровни описаний – по вертикали).

***Принцип декомпозиции*** (блочности) означает разбиение представлений каждого уровня на ряд составных частей (блоков) с возможностью раздельного (поблочного)

проектирования подсистем на уровне 1, объектов на уровне 2 и т.д., то есть каждый уровень разбивается на блоки.

Если решение задач высоких иерархических уровней предшествует решению задач более низких иерархических уровней, то проектирование называют ***нисходящим***. Если раньше выполняются этапы, связанные с низшими иерархическими уровнями, то проектирование называют ***восходящим***. У каждого из этих двух видов проектирования имеются преимущества и недостатки.

При нисходящем проектировании система разрабатывается в условиях, когда ее элементы еще не определены и, следовательно, сведения об их возможностях и свойствах носят предположительный характер.

При восходящем проектировании, наоборот, элементы проектируются раньше системы, и, следовательно, предположительный характер имеют требования к системе. В обоих случаях из-за отсутствия исчерпывающей исходной информации имеют место отклонения от возможных оптимальных технических результатов.

Поскольку принимаемые предположения могут не оправдаться, часто требуется повторное выполнение проектных процедур предыдущих этапов после выполнения проектных процедур последующих этапов. Такие повторения обеспечивают последовательное приближение к оптимальным результатам и обуславливают ***итерационный*** характер проектирования. Следовательно, итерационность нужно относить к важным принципам проектирования сложных систем.

На практике обычно сочетают восходящее и нисходящее проектирование. Например, восходящее проектирование имеет место на всех тех иерархических уровнях, на которых используются ***унифицированные*** (стандартные) элементы. Очевидно, что унифицированные элементы, ориентированные на применение в ряде различных систем определенного класса, разрабатываются раньше, чем та или иная конкретная система из этого класса. Использование типовых и унифицированных проектных решений приводит так же к упрощению и ускорению проектирования.

Однако, унификация целесообразна только в таких классах систем, в которых из сравнительно небольшого числа разновидностей элементов предстоит проектирование и изготовление большого числа систем. Именно эти разновидности элементов подлежат унификации.

Для сложных систем и для элементов, реализующих новые физические принципы или технологические возможности, в каждом конкретном случае приходится заново выполнять многоуровневое иерархическое проектирование.

В этих условиях целесообразно ставить вопрос не об унификации изделий, а об унификации средств их проектирования, в частности об унификации проектных процедур в рамках систем автоматизированного проектирования (САПР).

Окончательное описание проектируемой системы представляет собой полный комплект схемной, конструкторской и технологической документации, оформленной в соответствии с требованиями ГОСТов: ЕСКД (единая система конструкторской документации), ЕСТД (единая система технологической документации), ЕСПД (единая система программной документации). Этот комплект документации предназначен для использования в процессе реализации проекта системы.

Важное значение в этих описаниях имеют математические модели объектов проектирования, так как выполнение проектных процедур при автоматизированном проектировании основано на оперировании математическими моделями.

***Математическая модель (ММ) технического объекта*** – система математических объектов (чисел, переменных, матриц, множеств и т.п.) и отношений между ними, отражающих некоторые свойства технического объекта.

При проектировании используют математические модели, отражающие свойства объекта, существенные с позиции проектировщика.

Среди свойств объекта, отражаемых в описаниях на определенном иерархическом уровне, в том числе в ММ, различают свойства: систем; элементов систем и внешней среды, в которой должен функционировать объект. Количественное выражение этих свойств осуществляется с помощью величин, называемых ***параметрами***. Величины, характеризующие свойства системы, элементов системы и внешней среды, называют соответственно входными/выходными, внутренними и внешними параметрами.

Однако существование ММ не означает, что она известна разработчику и может быть представлена в явном функциональном виде. Типичной является ситуация, когда математическое описание процессов в проектируемой системе задается моделью в форме системы уравнений, в которой фигурирует вектор фазовых переменных. Фазовые переменные характеризуют физическое или информационное состояние объекта, а их изменения во времени выражают переходные процессы в объекте. Например, состояние некоторой фирмы можно определить такими фазовыми переменными: сырье, материалы, финансовые и трудовые ресурсы.

Выделим следующие особенности параметров в моделях проектируемых объектов:

1) Внутренние параметры в моделях $k$-го иерархического уровня становятся выходными параметрами в моделях более низкого ($k$+1)-го иерархического уровня. Так, например, трудовые ресурсы являются внутренними при проектировании производственной фирмы и в то же время выходными при проектировании отдела кадров этой фирмы.

2) Выходные параметры или фазовые переменные, фигурирующие в модели одной из подсистем (в одном из аспектов описаний), часто оказываются внешними параметрами в описании других подсистем (других аспектов). Так, например, выходные параметры подсистемы планирования выпуска продукции некоторой компании являются внешними параметрами подсистемы материально-технического снабжения этой компании.

3) Большинство выходных параметров объекта являются функционалами.

4) В техническом задании на проектирование должны фигурировать величины, называемые техническими требованиями к выходным параметрам (нормами выходных параметров). Данные нормы представляют собой границы допустимых диапазонов изменения выходных параметров.

### 2.3 Классификация типовых проектных процедур

Проектная процедура называется ***типовой***, если она предназначена для многократного применения при проектировании многих типов систем.

Различают проектные процедуры анализа и синтеза. Синтез заключается в создании описания системы, а анализ – в определении свойств и исследовании работоспособности системы по ее описанию, т.е. при синтезе создаются, а при анализе оцениваются проекты систем.

Процедуры анализа делятся на процедуры одно- и многовариантного анализа. При одновариантном анализе заданы значения внутренних и внешних параметров,

требуется определить значения выходных параметров системы. Подобная задача обычно сводится к однократному решению уравнений, составляющих математическую модель, что и обуславливает название этого вида анализа. Многовариантный анализ заключается в исследовании свойств системы в некоторой области пространства внутренних переменных. Такой анализ требует многократного решения систем уравнений (многократного выполнения одновариантного анализа).

Процедуры синтеза делятся на процедуры структурного и параметрического синтеза. Целью структурного синтеза является определение структуры системы – перечня типов элементов, составляющих систему, и способа связи элементов между собой в составе этой системы. Параметрический синтез заключается в определении числовых значений параметров элементов при заданных структуре и условиях работоспособности на выходные параметры системы, т.е. при параметрическом синтезе нужно найти точку или область в пространстве внутренних параметров, в которых выполняются те или иные условия (обычно условия работоспособности).

На рис. 2.2 представлена типичная последовательность проектных процедур на одном из этапов нисходящего проектирования. На предыдущем этапе решались задачи ($k$-1)-го иерархического уровня.

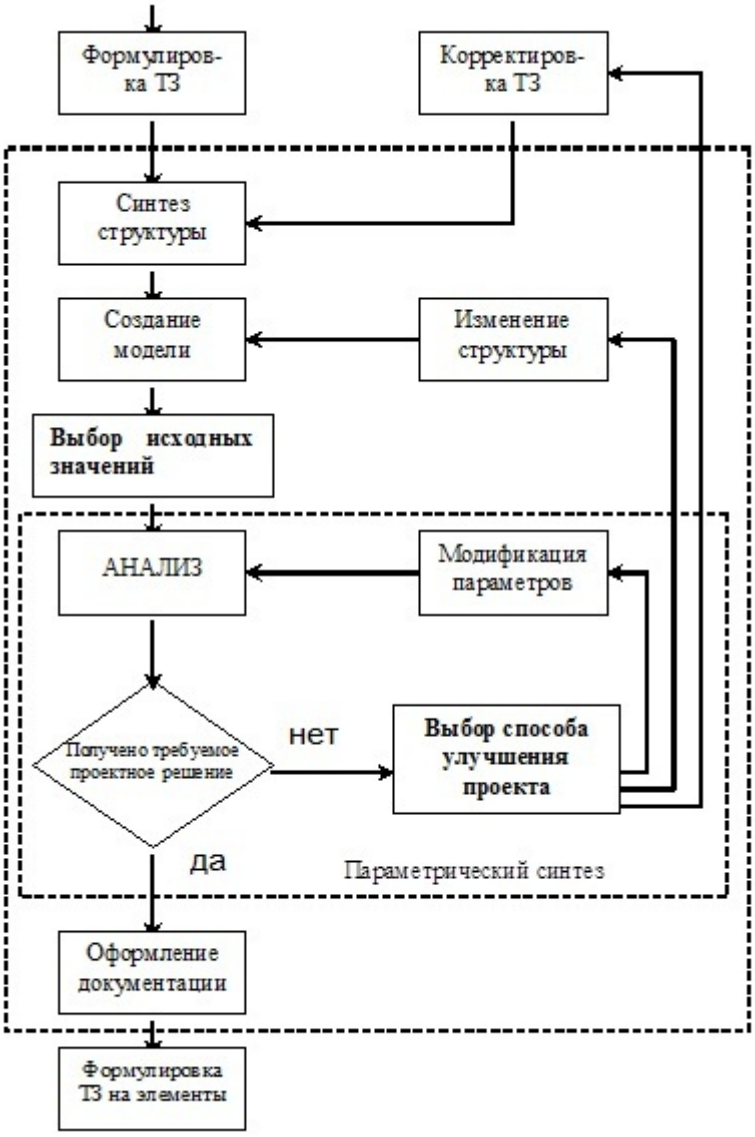


Рис. 2.2 – Схема процесса проектирования

Проектирование системы начинается с синтеза исходного варианта ее структуры. Для оценки этого варианта создается ее математическая модель.

После выбора исходных значений параметров элементов выполняется анализ варианта, по результатам которого становится возможной его оценка. Обычно оценка заключается в проверке выполнения условий работоспособности, сформулированных в техническом задании (ТЗ).

Если условия работоспособности выполняются в должной мере, то полученное проектное решение принимается, затем описывается система ($k$+1)-го уровня в принятой форме и формулируется ТЗ на проектирование элементов данного уровня. Если же полученное проектное решение неудовлетворительно, то выбирается один из возможных путей улучшения проекта.

Обычно проще всего изменить числовые значения параметров элементов. Совокупность процедур модификации параметров элементов, анализа и оценки результатов анализа представляет собой процедуру ***параметрического синтеза***. Если

модификации параметров целенаправленны и подчинены стратегии поиска наилучшего значения некоторого показателя качества, то процедура параметрического синтеза является ***процедурой оптимизации***.

Возможно, что путем параметрического синтеза не удастся добиться приемлемой степени выполнения условий работоспособности. Тогда используют другой путь, связанный с модификациями структуры. Новый вариант структуры синтезируется, для него повторяются процедуры формирования модели и параметрического синтеза.

Если не удается получить приемлемое решение и на этом пути, то ставится вопрос о корректировке ТЗ, сформулированного на предыдущем этапе проектирования. Такая корректировка может потребовать повторного выполнения ряда процедур $k$-го иерархического уровня, что и обуславливает итерационный характер проектирования.

Существует характерная особенность взаимосвязи проектных процедур анализа и синтеза. Эта взаимосвязь имеет характер вложенности процедуры анализа в процедуру оптимизации (параметрического синтеза) и процедуры оптимизации в процедуру синтеза (структурного и параметрического).

Вложенность означает, что:

– анализ входит как составная часть в оптимизацию, а оптимизация – в синтез;

– однократное выполнение процедуры оптимизации требует многократного выполнения процедуры анализа;

– однократное решение задачи синтеза требует многократного решения задачи оптимизации.

Очевидно, такой же характер взаимодействия имеют процедуры анализа: однократный многовариантный анализ основан на многократном одновариантном анализе.

Нетрудно подсчитать, что синтез проектного решения может потребовать чрезмерно большого количества вариантов анализа. Один из путей решения этой проблемы – применение достаточно точных и сложных математических моделей и алгоритмов анализа только на завершающих итерациях синтеза. Для большинства просматриваемых вариантов структуры при этом выполняется лишь ориентировочная оценка на основе косвенных критериев, упрощенных моделей и алгоритмов. Такая оценка позволит без существенных затрат вычислительных ресурсов отсеять большинство неперспективных вариантов и оставить для тщательного анализа малое число вариантов.

## Контрольные вопросы ко 2 главе:

1. Что такое проект информационной системы?
2 Что понимается под процессом проектирования информационной системы?
3. Что является объектами проектирования информационной системы?
4. Кто выступает в качестве субъекта проектирования информационной системы?
5. Что такое технология проектирования?
6. Какие основные требования предъявляются к выбираемой технологии проектирования?
7. Что составляет основу технологии проектирования информационной системы?
8. Какие принципы используются при проектировании сложных систем?
9. Какие элементы называются базовыми?
10. Какое проектирование называют нисходящим, а какое – восходящим?
11. Что представляет собой итерационный характер проектирования?

12. Какие элементы называются унифицированными?
13. Что такое математическая модель технического объекта?
14. Какие величины называются параметрами модели?
15. Какие переменные называются фазовыми в моделях объекта?
16. В чем заключаются особенности параметров в моделях проектируемых объектов?
17. Какая проектная процедура называется типовой?
18. Какие бывают проектные процедуры?
19. На какие процедуры делятся процедуры анализа?
20. В чем заключается смысл процедуры одновариантного анализа?
21. В чем смысл процедуры многовариантного анализа?
22. На какие процедуры делятся процедуры синтеза?
23. В чем заключается смысл процедуры структурного синтеза?
24. В чем заключается смысл процедуры параметрического синтеза?
25. Какая процедура называется процедурой оптимизации?

# 3 Структурный подход к проектированию информационных систем

## 3.1 Сущность структурного подхода

Сущность структурного подхода к разработке ИС заключается в ее декомпозиции на автоматизируемые функции: т.е. система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, эти подфункции подразделяются на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны [4].

Все наиболее распространенные методологии структурного подхода базируются на ряде общих принципов:

– принцип решения сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;

– принцип иерархического упорядочивания, т.е. принцип организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

– принцип абстрагирования: заключается в выделении существенных аспектов системы и отвлечения от несущественных;

– принцип формализации: заключается в необходимости строгого методического подхода к решению проблемы;

– принцип непротиворечивости: заключается в обоснованности и согласованности элементов;

– принцип структурирования данных: заключается в том, что данные должны быть структурированы и иерархически организованы.

На стадии проектирования ИС модели расширяются, уточняются и дополняются диаграммами, отражающими структуру программного обеспечения: архитектуру ПО, структурные схемы программ и диаграммы экранных форм.

Перечисленные модели в совокупности дают полное описание ИС независимо от того, является ли она существующей или вновь разрабатываемой. Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описания системы.

## 3.2 Методология функционального моделирования IDEF0

Методология IDEF0 использует технологию SADT (Structured Analysis and Design Technique) и представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная IDEF0-модель отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями [4].

IDEF0-модель дает полное, точное и адекватное описание системы, имеющее конкретное назначение. Целью модели является получение ответов на некоторую совокупность вопросов. Эти вопросы неявно присутствуют (подразумеваются) в процессе анализа и, следовательно, они руководят созданием модели и направляют его. Это означает, что сама модель должна будет дать ответы на эти вопросы с заданной степенью точности. Если модель отвечает не на все вопросы или ее ответы недостаточно точны, то мы говорим, что модель не достигла своей цели. Определяя модель таким образом, IDEF0 закладывает основы практического моделирования. Только поняв,

насколько хорошо нужно ответить на поставленные вопросы, можно определить, когда процесс моделирования можно считать завершенным (т.е. когда модель будет соответствовать поставленной цели).

Модель является некоторым толкованием системы. Поэтому субъектом моделирования служит сама система. Однако моделируемая система никогда не существует изолированно: она всегда связана с окружающей средой. Причем зачастую трудно сказать, где кончается система и начинается среда. По этой причине в методологии IDEF0 подчеркивается необходимость точного определения границ системы. IDEF0-модель всегда ограничивает свой субъект, т.е. модель устанавливает точно, что является и что не является субъектом моделирования, описывая то, что входит в систему, и подразумевая то, что лежит за ее пределами. Ограничивая субъект, IDEF0-модель помогает сконцентрировать внимание именно на описываемой системе и позволяет избежать включения посторонних субъектов. Вот почему утверждается, что IDEF0-модель должна иметь единственный субъект.

С определением модели тесно связана позиция, с которой наблюдается система и создается ее модель. Поскольку качество описания системы резко снижается, если оно не сфокусировано ни на чем, IDEF0 требует, чтобы модель рассматривалась все время с одной и той же позиции. Эта позиция называется «точкой зрения» данной модели. «Точку зрения» лучше всего представлять себе как место (позицию) человека или объекта, в которое надо встать, чтобы увидеть систему в действии. С этой фиксированной точки зрения можно создать согласованное описание системы так, чтобы в модели не смешивались бы несвязанные описания.

После того как определены субъект, цель и точка зрения модели, начинается первая интеграция процесса моделирования по методологии IDEF0. Субъект определяет, что включить в модель, а что исключить из нее. Точка зрения диктует автору модели выбор нужной информации о субъекте и форму ее подачи. Цель становится критерием окончания моделирования. Конечным результатом этого процесса является набор тщательно взаимоувязанных описаний, начиная с описания самого верхнего уровня всей системы и завершая подробным описанием функций или операций системы.

Результатом применения методологии IDEF0 является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга. Диаграммы – главные компоненты модели, все функции ИС и интерфейсы на них представлены как блоки и дуги.

Блок описывает функцию. Внутри каждого блока помещаются его имя и номер. Имя должно быть активным глаголом или частью глагольной формы, описывающим функцию. Номер блока размещается в правом нижнем углу (рис. 3.1). Номера блоков используются для их идентификации на диаграмме и в соответствующем тексте.


Рисунок 3.1 – Функциональный блок в нотации IDEF0

Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация входит в блок сверху, в то время как информация, которая подвергается обработке, показывается с левой стороны блока, а результаты выхода – с правой стороны. Механизм (человек или автоматизированная система), который осуществляет операцию, представляется дугой, входящей в блок снизу (рис. 3.2).

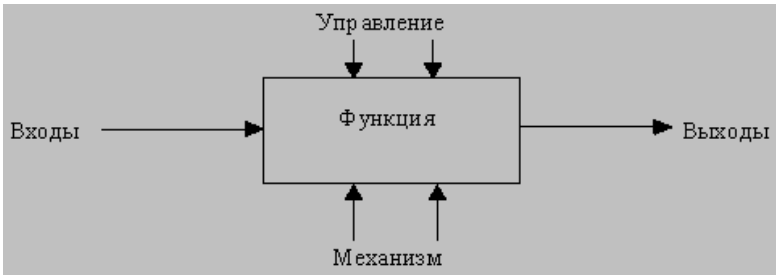


Рисунок 3.2 – Функциональный блок и интерфейсные дуги в нотации IDEF0

Одной из наиболее важных особенностей методологии IDEF0 является постепенное введение все больших уровней детализации по мере создания диаграмм, отображающих модель.

На рис. 3.3 приведены четыре диаграммы и их взаимосвязи, показана структура IDEF0-модели. Каждый компонент модели может быть декомпозирован на другой диаграмме. Каждая диаграмма иллюстрирует «внутреннее строение» блока на родительской диаграмме. Дуги, входящие в блок и выходящие из него на диаграмме верхнего уровня, являются точно теми же самыми, что и дуги, входящие в диаграмму нижнего уровня и выходящие из нее, потому что блок и диаграмма представляют одну и ту же часть системы.

Построение IDEF0-модели начинается с представления всей системы в виде простейшей компоненты – одного блока и дуг, изображающих интерфейсы с функциями вне системы. Поскольку единственный блок представляет всю систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг – они также представляют полный набор внешних интерфейсов системы в целом.

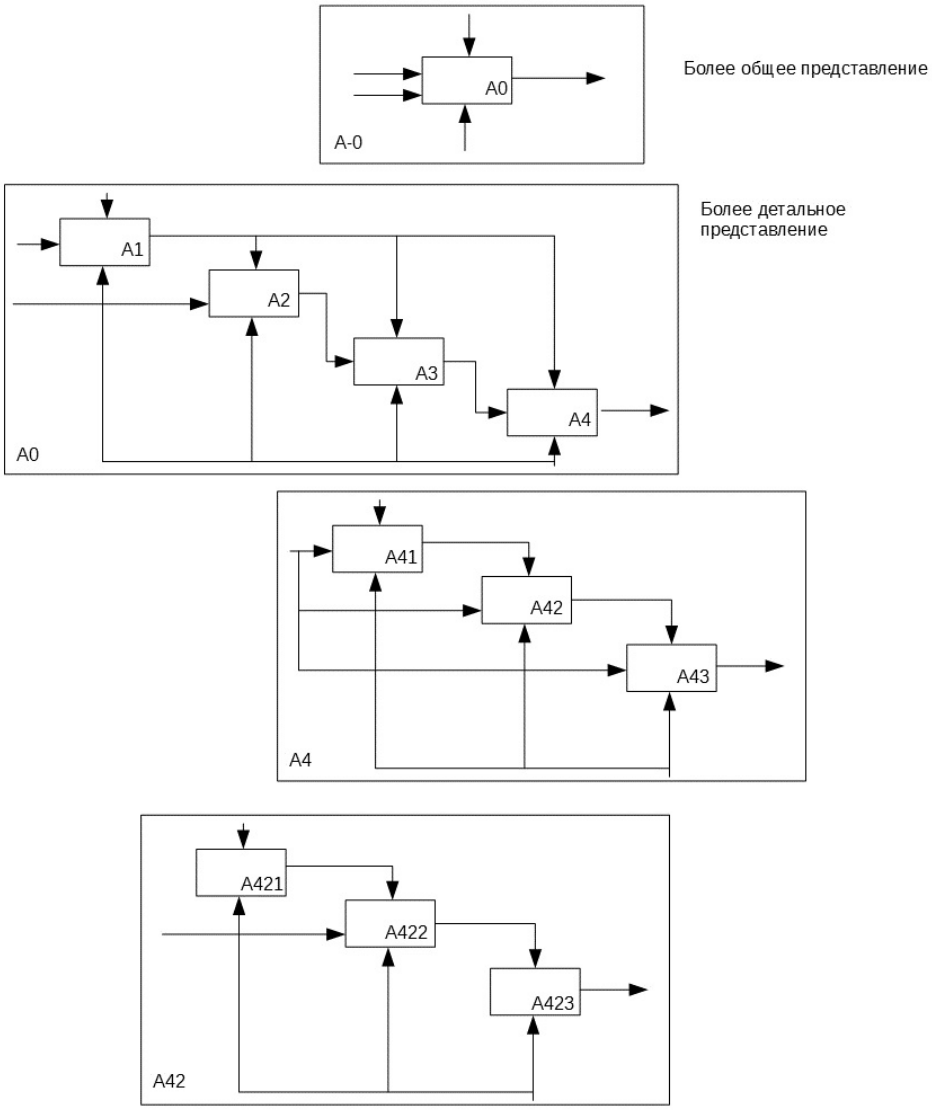


Рисунок 3.3 – Структура IDEF0-модели. Декомпозиция диаграмм

Затем блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами. Эти блоки представляют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых представлена как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом для более детального представления. Во всех случаях каждая подфункция может содержать только те элементы, которые входят в исходную функцию.

Модель IDEF0 представляет собой серию диаграмм с сопроводительной документацией, разбивающих сложный объект на составные части, которые представлены в виде блоков. Детали каждого из основных блоков показаны в виде блоков на других диаграммах. Каждая детальная диаграмма является декомпозицией блока из более общей диаграммы. На каждом шаге декомпозиции более общая диаграмма называется ***родительской*** для более детальной диаграммы.

Некоторые дуги присоединены к блокам диаграммы обоими концами, у других же один конец остается не присоединенным. Не присоединенные дуги соответствуют входам, управлениям и выходам родительского блока. Источник или получатель этих пограничных дуг может быть обнаружен только на родительской диаграмме.

Каждый блок на диаграмме имеет свой номер. Блок любой диаграммы может быть далее описан диаграммой нижнего уровня, которая, в свою очередь, может быть далее детализирована с помощью необходимого числа диаграмм. Таким образом, формируется иерархия диаграмм.

Для того, чтобы указать положение любой диаграммы или блока в иерархии, используются иерархические номера диаграмм. Например, А21 является диаграммой, которая детализирует блок 1 на диаграмме А2. Аналогично, А2 детализирует блок 2 на диаграмме А0. Номером самого верхнего блока является А-0.

Блоки IDEF0 никогда не размещаются на диаграмме случайным образом. Они размещаются по степени важности, как ее понимает автор диаграммы. В IDEF0 этот относительный порядок называется ***доминированием***. Доминирование понимается как влияние, которое один блок оказывает на другие блоки диаграммы. Например, самым доминирующим блоком диаграммы может быть либо первый из требуемой последовательности функций, либо планирующая или контролирующая функция, влияющая на все другие функции. Наиболее доминирующий блок обычно размещается в верхнем левом углу диаграммы, а наименее доминирующий - в правом нижнем углу. В результате получается «ступенчатая» схема, подобная представленной на рисунке 3.3. Расположение блоков на диаграмме отражает авторское определение доминирования. Таким образом, топология диаграммы показывает, какие функции оказывают большее влияние на остальные. Чтобы подчеркнуть это, IDEF0-аналитик может перенумеровать блоки в соответствии с порядком их доминирования.

В методологии IDEF0 требуется только пять типов взаимосвязей между блоками для описания их отношений: управление, вход, обратная связь по управлению, обратная связь по входу, выход-механизм. Связи по управлению и входу являются простейшими, поскольку они отражают прямые воздействия, которые интуитивно понятны и очень просты.

Отношение управления возникает тогда, когда выход одного блока непосредственно влияет на блок с меньшим доминированием (рис. 3.4). Отношение входа возникает тогда, когда выход одного блока становится входом для блока с меньшим доминированием (рис. 3.5).

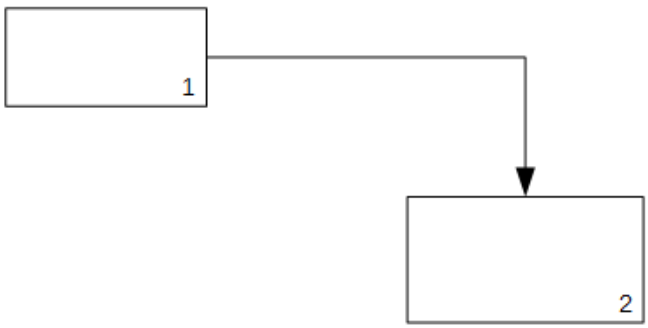


Рисунок 3.4 – Отношение управления между функциональными блоками в стандарте IDEF0

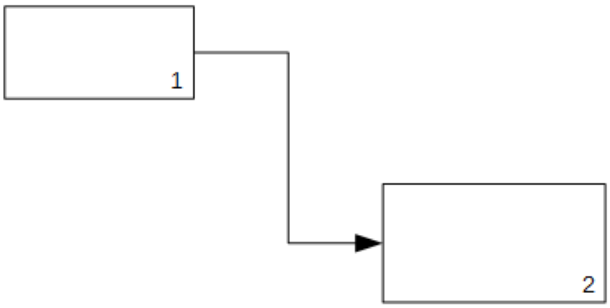


Рисунок 3.5 – Отношение входа между функциональными блоками в стандарте IDEF0

Обратная связь по управлению и обратная связь по входу являются более сложными, поскольку они представляют итерацию или рекурсию, т.е. выходы из одной функции влияют на будущее выполнение других функций, что впоследствии влияет на исходную функцию. Обратная связь по управлению возникает тогда, когда выход некоторого блока влияет на блок с большим доминированием (рис. 3.6). Входная обратная связь имеет место тогда, когда выход одного блока становится входом другого блока с большим доминированием (рис. 3.7).

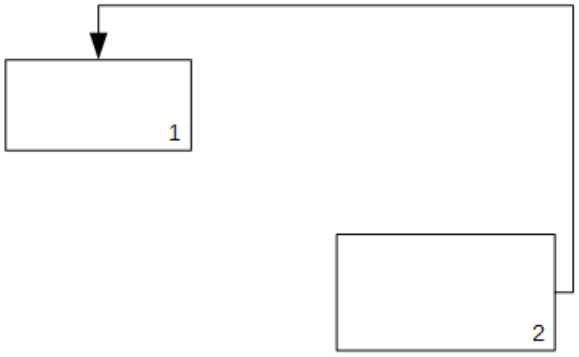


Рисунок 3.6 – Отношение обратной связи по управлению между функциональными блоками в стандарте IDEF0

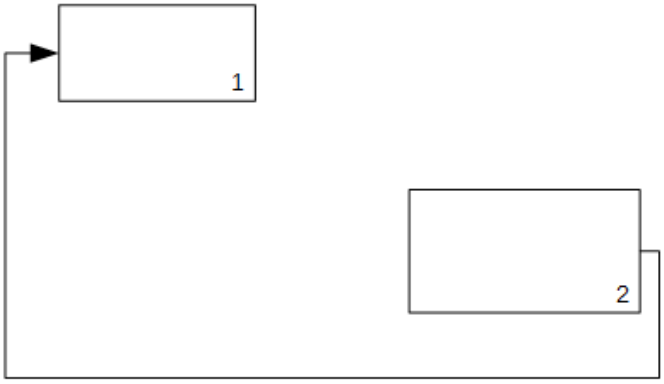


Рисунок 3.7 – Отношение обратной связи по входу между функциональными блоками в стандарте IDEF0

Связи «выход-механизм» встречаются нечасто и представляют особый интерес. Они отражают ситуацию, при которой выход одной функции становится средством достижения цели для другой (рис. 3.8). Связи «выход-механизм» характерны при распределении источников ресурсов (например, требуемые инструменты, обученный персонал, физическое пространство, оборудование, финансирование, материалы).

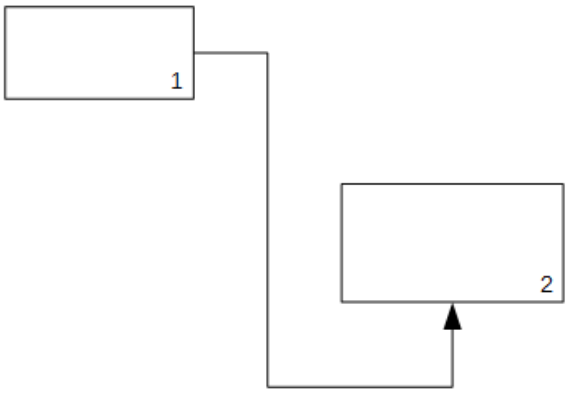


Рисунок 3.8 – Связь «выход-механизм» между функциональными блоками в стандарте IDEF0

Разветвления дуг, изображаемые в виде расходящихся линий (рис. 3.9), означают, что все содержимое дуг или их часть может появиться в каждом ответвлении дуги. Дуга всегда помечается до разветвления, чтобы дать название всему набору.


Рисунок 3.9 – Отображение разветвления дуги в стандарте IDEF0

Кроме того, каждая ветвь дуги может быть помечена или не помечена в соответствии со следующими правилами:

– непомеченные ветви содержат все объекты, указанные в метке дуги перед разветвлением (т.е. все объекты принадлежат этим ветвям);

– ветви, помеченные после точки разветвления, содержат все объекты или их часть, указные в метке дуги перед разветвлением (т.е. каждая метка ветви уточняет, что именно содержит ветвь).

Слияние дуг в IDEF0, изображаемое как сходящиеся вместе линии (рис. 3.10), указывает, что содержимое каждой ветви идет на формирование метки для дуги, являющейся результатом слияния исходных дуг. После слияния результирующая дуга всегда помечается для указания нового набора объектов, возникшего после объединения.


Рисунок 3.10 – Отображение слияния дуги в стандарте IDEF0

Кроме того, каждая ветвь перед слиянием может помечаться или не помечаться в соответствии со следующими правилами:

– непомеченные ветви содержат все объекты, указанные в общей метке дуги после слияния (т.е. все объекты исходят из всех ветвей);

– помеченные перед слиянием ветви содержат все или некоторые объекты из перечисленных в общей метке после слияния (т.е. метка ветви ясно указывает, что содержит ветвь).

Хорошая методология структурного анализа, позволяющая создавать отдельные диаграммы, должна гарантировать правильное соединение всех диаграмм для

образования согласованной модели. IDEF0-диаграммы имеют внешние дуги – дуги, как бы выходящие наружу и ведущие к краю страницы. Эти дуги являются интерфейсом между диаграммой и остальной частью модели. IDEF0 требует, чтобы все внешние дуги диаграммы были согласованы с дугами, образующими границу этой диаграммы. Другими словами, диаграмма должна быть «состыкована» со своей родительской диаграммой. Обычно это означает, что внешние дуги согласованы по числу и наименованию (но не обязательно по расположению) с дугами, касающимися декомпозированного блока родительской диаграммы.

В IDEF0 принята система обозначений, позволяющая аналитику точно идентифицировать и проверять связи по дугам между диаграммами. Эта схема кодирования дуг «ICOM» получила название по первым буквам английских эквивалентов слов вход (Input), управление (Control), выход (Output), механизм (Mechanism). Коды ICOM чрезвычайно эффективны, поскольку они позволяют аналитику быстро проверять согласованность внешних дуг диаграммы с граничными дугами соответствующего блока родительской диаграммы.

Они также обеспечивают согласованность декомпозиции, поскольку все дуги, входящие в диаграмму и выходящие из нее, должны быть учтены. Одним из способов такой стыковки может служить присваивание кодов ICOM внешним дугам новой диаграммы согласно следующим правилам:

1) Представьте себе рисунок новой диаграммы внутри разлагаемого блока. Продлите внешние дуги почти до края диаграммы. Зрительно соедините каждую внешнюю дугу диаграммы с соответствующей граничной дугой декомпозируемого блока.

2) Присвойте код каждой зрительной связи. Используйте I для входных дуг, C – для связей между дугами управления, O – для связей между выходными дугами, M – для связей между дугами механизма.

3) Добавьте после каждой буквы цифру, соответствующую положению данной дуги среди других дуг того же типа, касающихся родительского блока. Причем входные и выходные дуги пересчитываются сверху вниз, а дуги управлений и механизмов пересчитываются слева направо. Теперь запишите каждый код около окончания каждой внешней дуги, например: I1, I2, O3, C1, M2.

## Контрольные вопросы к 3 главе

1. Какие основные составляющие методологии IDEF0?
2. Что из себя представляет функциональный блок?
3. В чем заключается цель IDEF0-модели?
4. Что является субъектом моделирования в IDEF0-модели?
5. Что такое «точка зрения модели»?
6. Как взаимосвязаны диаграммы IDEF0-модели между собой?
7. Есть ли связь между функциональным блоком и диаграммы?
8. Какие диаграммы называются родительскими?
9. Как нумеруются диаграммы и блоки?
10. Что такое доминирование IDEF0-модели?
11. Какие типы взаимосвязей между блоками существуют?
12. Как формируются ICOM-коды?

# 4 Объектно-ориентированный подход к проектированию информационных систем

## 4.1 Общие сведения об объектно-ориентированном проектировании информационных систем

Объектно-ориентированное проектирование — это подход к решению задач с использованием моделей, основанных на понятиях реального мира [5]. Фундаментальным элементом является объект, объединяющий структуру данных с поведением. Характеристики объектно-ориентированного подхода включают в себя индивидуальность, классификацию, наследование и полиморфизм.

***Индивидуальность*** означает, что данные делятся на дискретные сущности, хорошо отличимые друг от друга. Эти сущности называются объектами. Объекты могут быть конкретными, как, например, файл в составе файловой системы, или концептуальными, как политика планирования в многопроцессорной операционной системе. Каждый объект обладает своей собственной внутренней индивидуальностью. Два объекта различимы даже в том случае, если значения всех их атрибутов (таких как имя или размер) одинаковы.

***Классификация*** означает, что объекты с одинаковыми структурами данных (атрибутами) и поведением (операциями) группируются в классы. Класс — это абстракция, описывающая свойства, важные для конкретного приложения, и игнорирующая все остальное. Любой выбор классов произволен и зависит от приложения. Каждый класс описывает множество индивидуальных объектов, которое может быть бесконечным. Каждый объект из этого множества называется экземпляром класса. Объект имеет свои собственные значения атрибутов, но названия атрибутов и операций являются общими для всех экземпляров класса.

***Наследование*** – это наличие у разных классов, образующих иерархию, общих атрибутов и операций. Суперкласс задает наиболее общую информацию, которую затем уточняют и улучшают его подклассы. Каждый подкласс соединяет в себе, то есть наследует, все черты его суперкласса, к которым добавляет собственные уникальные черты. Подклассам необязательно воспроизводить все черты суперкласса. Возможность выделять общие черты нескольких классов в суперкласс значительно сокращает количество повторений в проектах и программах и является одним из основных достоинств объектно-ориентированной технологии.

***Полиморфизм*** означает, что одна и та же операция может подразумевать разное поведение в разных классах. Операция — это процедура или трансформация, которую объект выполняет сам или которая осуществляется с данным объектом. Реализация операций в конкретном классе называется методом. Поскольку объектно-ориентированная операция является полиморфной, в разных классах объектов она может быть реализована разными методами.

В реальном мире операция является абстрагированием похожего поведения у объектов одного рода. Каждый объект «сам знает», как выполнить свои собственные операции. В объектно-ориентированных языках программирования выбор подходящего метода для реализации операции осуществляется автоматически, исходя из имени операции и класса объекта, к которому она относится. Клиенту операции нет необходимости знать о том, сколько еще методов могут реализовывать данную полиморфическую операцию. Разработчики могут добавлять новые классы, не меняя

существующий код, при условии, что они предоставляют методы для каждой возможной операции.

Объектно-ориентированное проектирование — это концептуальный процесс, независимый от языка программирования, по крайней мере, до последних этапов. Фактически это образ мышления, а не методика программирования. Главное преимущество объектно-ориентированного проектирования состоит в том, что оно помогает тем, кто пишет спецификации, разработчикам и заказчикам ясно выражать абстрактные концепции и обсуждать их друг с другом. Таким образом, облегчается составление спецификаций, анализ, документирование и определение интерфейсов и, конечно же, программирование.

Объектно-ориентированный подход состоит из построения модели приложения и последующей ее детализации. Одна и та же система обозначений используется на всем протяжении процесса разработки, начиная с анализа и заканчивая реализацией. Информация, добавленная на одном из этапов, не будет утеряна или даже преобразована при переходе к следующему этапу. Описываемая методология включается в себя следующие этапы:

1. Концептуализация системы. Разработка программного обеспечения начинается с бизнес-аналитиков или пользователей, которые придумывают приложение и формируют первичные требования к нему.

2. Анализ. Аналитик тщательно исследует и переформулирует требования, конструируя модели, исходя из концепций системы. Аналитик должен работать с заказчиком, чтобы добиться понимания задачи, потому что формулировки редко оказываются полными и корректными. Аналитическая модель — это сжатая и точная абстракция того, что именно должна сделать система (а не то, каким образом это будет сделано). Аналитическая модель не должна содержать никаких решений относительно реализации. Аналитическая модель состоит из двух частей: модели предметной области — описания объектов реального мира, отражаемых системой, и модели приложения — описания видимых пользователю частей самого приложения. Например, для приложения биржевого маклера объектами предметной области могут быть акции, облигации, торги и комиссия. Объекты модели приложения могут управлять выполнением торгов и отображать результаты. Хорошая модель должна быть доступной для понимания и критики со стороны экспертов, не являющихся программистами.

3. Проектирование системы. Команда разработчиков продумывает стратегию решения задачи на высшем уровне, определяя архитектуру системы. На этом этапе определяются политики, которые послужат основой для принятия решений на следующих этапах. Проектировщик системы должен выбрать параметры системы, по которым будет проводиться оптимизация, предложить стратегический подход к задаче, провести предварительное распределение ресурсов. Например, проектировщик может решить, что любые изменения изображения на экране рабочей станции должны быть быстрыми и плавными, даже при перемещении и закрытии окон. На основании этого решения он может выбрать подходящий протокол обмена и стратегию буферизации памяти.

4. Проектирование классов. Проектировщик классов уточняет аналитическую модель в соответствии со стратегией проектирования системы. Он прорабатывает объекты предметной области и объекты модели приложения, используя одинаковые объектно-ориентированные концепции и обозначения, несмотря на то что эти объекты лежат в разных концептуальных плоскостях. Цель проектирования классов состоит в

том, чтобы определить, какие структуры данных и алгоритмы требуются для реализации каждого класса.

5. Реализация. Ответственные за реализацию занимаются переводом классов и отношений, образовавшихся на предыдущем этапе, на конкретный язык программирования, воплощением их в базе данных или аппаратном обеспечении. Никаких усложнений на этом этапе быть не должно, потому что все ответственные решения уже были приняты на предыдущих этапах. В процессе реализации необходимо использовать технологии разработки программного обеспечения, чтобы соответствие кода проекту было очевидным, а система оставалась гибкой и расширяемой.

Объектно-ориентированные концепции действуют на протяжении всего жизненного цикла информационной системы. Одни и те же классы будут переходить от одного этапа к другому без всяких изменений в нотации, хотя на последних этапах они существенно обрастут деталями. Концепции индивидуальности, классификации, полиморфизма и наследования действуют на протяжении всего процесса разработки.

Некоторые классы не входят в аналитическую модель. Они появляются позднее, на этапах проектирования и реализации. Например, структуры данных, подобные деревьям, хэш-таблицам и связным спискам, редко появляются в реальном мире и обычно невидимы пользователям. Проектировщики добавляют их в систему для того, чтобы обеспечить поддержку выбранных алгоритмов. Объекты структур данных существуют внутри компьютера и не являются непосредственно наблюдаемыми.

Тестирование не рассматривается как отдельный этап. Тестирование очень важно, но оно должно быть частью системы контроля качества, которая применяется на протяжении всего жизненного цикла. Разработчики должны сравнивать аналитические модели с реальностью. Они должны проверять проектировочные модели на наличие ошибок различных видов, а не только тестировать корректность реализации. Выделение всех операций по контролю качества в отдельный этап стоит дороже и оказывается менее эффективно.

В 1996 году группа управления объектами (Object Management Group – OMG [6]) объявила конкурс на лучший стандарт обозначений для объектно-ориентированного моделирования. В этом конкурсе приняло участие несколько компаний. В результате их предложения были объединены в конечную систему. В ноябре 1997 года группа OMG приняла получившийся в результате унифицированный язык моделирования (UML – Unified Modeling Language) в качестве стандарта. Компании, принимавшие участие в конкурсе, передали права на UML группе OMG, которая стала владельцем торговой марки и спецификаций UML. Эта группа управляет дальнейшим совершенствованием языка UML. Система обозначений UML оказалась настолько удачной, что вытеснила практически все другие системы.

### 4.2 Объектно-ориентированные концепции

В объектно-ориентированной технологии широко используются несколько базовых концепций. Они не ограничиваются рамками объектно-ориентированных систем, но объектная ориентированность означает, прежде всего, поддержку этих концепций.

Абстракция. Абстракция означает сосредоточение на важнейших аспектах приложения и игнорирование всех остальных. Сначала принимается решение о том, что представляет собой объект и что он делает, а затем подбирается способ его реализации. Использование абстракций позволяет сохранить свободу принятия решений как можно

дольше благодаря тому, что детали не фиксируются раньше времени. Большинство современных языков программирования позволяют абстрагировать данные, на наследование и полиморфизм значительно расширяют возможности концепции абстрагирования. Умение создавать абстракции, вероятно, является самым важным качеством для объектно-ориентированного разработчика.

Инкапсуляция. Инкапсуляция, или, иначе говоря, сокрытие информации, состоит в отделении внешних аспектов объекта, доступных другим объектам, от деталей внутренней реализации, которые от других объектов скрываются. Инкапсуляция исключает возникновение взаимозависимости участков программы, из-за которых небольшие изменения приводят к значительным непредвиденным последствиям. Реализация объекта может быть изменена безо всяких последствий для использующих его приложений. Изменение реализации может быть предпринято для повышения производительности, устранения ошибки, консолидации кода или для подготовки к переносу программы на другие системы.

Объединение данных и поведения. При вызове операции не нужно беспокоиться о том, сколько реализаций этой операции существует в системе. Полиморфизм операторов перекладывает ответственность за выбор подходящей реализации с вызывающего кода на иерархию классов. Рассмотрим в качестве примера обычный (не объектно-ориентированный) код, отображающий содержимое окна. Такой код должен учитывать тип каждой фигуры (многоугольник, окружность, текст) и вызывать соответствующие типу процедуры. Объектно-ориентированный код будет вызывать для каждой фигуры метод «прорисовать». Каждый объект выберет подходящую процедуру согласно своему классу. Это облегчает поддержку программы, потому что добавление нового класса не требует изменения вызывающего кода.

Совместное использование. Объектно-ориентированные технологии способствуют совместному использованию сущностей на самых разных уровнях. Наследование структур данных вместе с поведением дает возможность подклассам совместно использовать общий код. Совместное использование при наследовании является одним из главных преимуществ объектно-ориентированных языков. Однако, концептуальная ясность, проистекающая из осознания того, что разные операции на деле представляют собой одно и то же, оказывается важнее, чем экономия кода. Благодаря этому сокращается количество различных ситуаций, которые требуется понять и проанализировать. Объектно-ориентированная разработка позволяет не только совместно работать с общей информацией внутри приложения, но и повторно использовать проекты и код в последующих программах. Объектно-ориентированные средства помогают строить библиотеки повторно используемых компонентов.

Выделение сущности объекта. Объектно-ориентированная технология выделяет то, чем объект *является*, а не то, как он *используется*. Использование объекта зависит от особенностей приложения и часто изменяется в процессе разработки. По мере уточнения требований, черты объекта остаются более стабильными, чем детали его использования, поэтому системы, основанные на объектной структуре, в конечном счете, оказываются более стабильными. При объектно-ориентированной разработке большее внимание уделяется структурам данным и меньшее – процедурам, нежели в методологиях, связанных с функциональным разбиением.

Когда целое больше суммы частей. Все объектно-ориентированные языки характеризуются поддержкой концепций индивидуальности, классификации, полиморфизма и наследования. Каждая из этих концепций может использоваться сама по себе, однако вместе они образуют нечто большее. Преимуществ объектно-

ориентированного подхода оказывается больше, чем может показаться с первого взгляда. Выделение основных свойств объекта заставляет разработчика более внимательно относиться к тому, чем объект является и что он делает. В результате система оказывается более ясной, универсальной и устойчивой, чем в том случае, если бы основное значение придавалось функциям и операциям.

Для описания системы с различных точек зрения используют три типа моделей. Модель классов описывает объекты, входящие в состав системы и отношения между ними. Модель состояний описывает историю жизни объектов. Модель взаимодействий описывает взаимодействия между объектами. Каждая модель применяется на всех этапах проектирования и постепенно обрастает деталями. Полное описание системы требует наличия всех трех моделей.

Модель классов описывает статическую структуру объектов системы и их отношения. Эта модель определяет контекст разработки программы, то есть предметную область. Модель классов изображается на диаграммах классов.

*Диаграмма классов* – это граф, вершинами которого являются классы, а ребрами – их отношения.

Модель состояний описывает изменяющиеся со временем аспекты объектов. Эта модель реализуется посредством диаграмм состояний.

*Диаграмма состояний* – это граф, вершинами которого являются состояния, а ребрами – переходы между состояниями, инициируемые событиями.

Модель взаимодействия описывает кооперацию объектов системы для достижения лучших результатов. Построенные модели начинаются с вариантов использования, которые затем уточняются на диаграммах последовательности и диаграммах деятельности. Вариант использования описывает функциональность системы, то есть то, что системы делает для пользователей. Диаграмма последовательности изображает взаимодействие объектов и временную последовательность этого взаимодействия. Диаграмма деятельности уточняет важные этапы обработки.

Три описанные модели являются связанными между собой составляющими полного описания системы. Центральной является модель классов, поскольку сначала нужно определить, *что* именно изменяется или трансформируется, а затем уже описывать, *когда* и *как* это происходит.

## 4.3 Моделирование классов

### 4.3.1 Концепции объекта и класса

Цель моделирования классов состоит в описании объектов.

*Объект* – это концепция, абстракция или сущность, обладающая индивидуальностью и имеющая смысл в рамках системы. Объекты часто бывают имена собственными или конкретными ссылками, которые используются в описании задач или при общении с пользователями. Некоторые объекты существуют или существовали в реальном мире (например, «Андрей»*)*, тогда как другие являются сугубо концептуальными сущностями (например, «тестовый прогон № 1234» или «формула корней квадратного уравнения»). Объекты третьего типа (например, «бинарное дерево» и «массив, связанный с переменной *a*»*)* добавляются в модель в процессе реализации и не имеют никакого отношения к физической реальности. Выбор объектов зависит от

природы задачи и от предпочтений разработчика. Корректное решение в данном случае не является единственным.

Все объекты обладают индивидуальностью и потому отличаются друг от друга. Два яблока одинакового цвета, формы и текстуры все равно являются разными яблоками. Вы можете съесть сначала одно из них, а потом второе. Похожие друг на друга близнецы тоже являются независимыми индивидуальностями. Индивидуальность означает, что объекты отличаются друг от друга внутренне, а не по внешним свойствам.

Объект является экземпляром класса. *Класс* описывает группу объектов с одинаковыми свойствами (атрибутами), одинаковым поведением (операциями), типами отношений и семантикой. В качестве примеров классов можно привести следующие: «человек», «компания», «процесс» и «окно». Каждый человек имеет имя и дату рождения, а также может где-либо работать. Каждый человек- процесс имеет владельца, приоритет и список необходимых ресурсов. Классы часто бывают именами нарицательными и именными группами, которые используются в описании задач или при общении с пользователями.

Объекты одного класса имеют одинаковые атрибуты и формы поведения. Большинство объектов отличаются друг от друга значениями своих атрибутов и отношениями с другими объектами. Однако, возможно существование разных объектов с одинаковыми значениями атрибутов, находящихся в одинаковых отношениях со всеми остальными объектами. Выбор классов зависит от природы и области применения приложения и зачастую является субъективным.

Объекты класса имеют общее семантическое значение, помимо обязательных общих атрибутов и поведения. Например, и амбар, и лошадь могут характеризоваться стоимостью и возрастом. С финансовой точки зрения они могли бы относиться к одному классу. Однако если разработчик учтет, что человек может красить амбар и кормить лошадь, то эти две сущности будут отнесены к разным классам. Интерпретация семантики зависит от назначения конкретного приложения и является делом субъективным.

Каждый объект «знает» свой собственный класс. Большинство объектно-ориентированных языков программирования позволяют определять класс объекта во время выполнения программы. Класс объекта – это его неявное свойство.

Если предметом моделирования являются объекты, то почему вообще заходит речь о классах? Все дело в необходимости абстрагирования. Группируя объекты в классы, мы производим абстрагирование в рамках задачи. Именно благодаря абстракциям моделирование оказывается столь мощным инструментом, позволяющим проводить обобщения от нескольких конкретных случаев к множеству подобных альтернатив. Общие определения (такие как название класса и названия атрибутов) хранятся отдельно для каждого класса, а не для каждого экземпляра.

Операции могут быть написаны один раз для целого класса, благодаря чему все объекты класса получают возможность повторно использовать написанный код. Например, все эллипсы имеют общие процедуры прорисовки, вычисления площади и проверки на наличие пересечения с некоторой прямой. У многоугольников будет свой набор процедур. Даже в особых случаях (например, для окружностей и квадратов) могут использоваться универсальные процедуры, хотя специальные алгоритмы могут быть и эффективнее.

Описания понятий объекта и класса не дает достаточной четкости и непригодны для описания сложных приложений. Необходимо средство описания моделей, которое

было бы согласованным, точным и простым в использовании. Модели структуры бывают двух типов: диаграммы классов и диаграммы объектов.

Диаграммы классов позволяют описать модель классов и их отношений (а значит и возможные объекты) при помощи графической системы обозначений. Диаграмма классов описывает бесконечное множество диаграмм объектов.

На рис. 4.1 показан класс (слева) и его экземпляры (справа). Объекты «Андрей», «Мария» и безымянная личность являются экземплярами класса «Чело-век». В языке UML для обозначения объекта используется прямоугольник, внутри которого ставится имя объекта, двоеточие и имя класса, к которому относится этот объект. Имя объекта и имя класса подчеркиваются. Также используется дополнительное соглашение, которое состоит в том, что имена объектов и классов выделяется полужирным шрифтом.

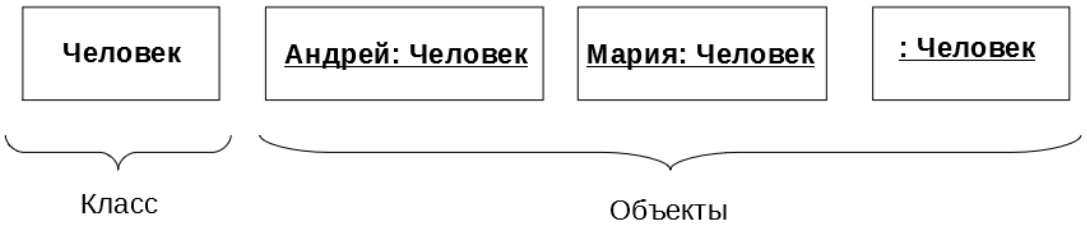


Рисунок 4.1 – Отображение объектов и классов в нотации UML

Для обозначения класса в UML тоже используется прямоугольник. Имя класса указывается полужирным шрифтом, располагаясь посередине прямоугольника, и начинается с заглавной буквы. Эти соглашения используются для ссылок на объекты, классы и другие конструкции. Соглашение об использовании заглавных букв между строчных популярно в литературе, посвященной объектно-ориентированным технологиям, но не является требованием языка UML.

Значение – это элемент данных. Значения можно определить, изучив примеры, приведенные в документации по поставленной задаче.

*Атрибут* – это именованное свойство класса, описывающее значение, которое может иметь каждый объект класса. Атрибуты получаются абстрагированием типичных значений. Можно привести следующую аналогию: объект относится к классу так, как значение относится к атрибуту. В моделях классов преобладают структурные конструкции. Атрибуты служат для уточнения характеристик классов и отношений.

Атрибутами объектов класса «Человек» являются, например, «имя», «датаРождения». У каждого конкретного объекта атрибут принимает свое конкретное значение. Например, у объекта Андрей атрибут «датаРождения» может иметь значение «21 октября 1983». У разных объектов один и тот же атрибут может иметь как разные, так и одинаковые значения. Имя атрибута уникально в рамках класса (но не обязательно уникально во множестве всех классов), поэтому у разных классов может быть атрибут с одним и тем же названием.

Не следует путать значения с объектами. Атрибут должен описывать значения, а не объекты. В отличие от объектов значения не обладают индивидуальностью. Например, все экземпляры числа 17 неразличимы между собой. Точно так же неразличимы экземпляры строкового значения «Россия». С другой стороны, страна, которая называется Россия, является объектом, у которого атрибут «название» имеет значение «Россия».

На рис. 4.2 показана система обозначений атрибутов. Класс «Человек» имеет атрибуты «имя» и «датаРождения». «имя» – это строковый тип данных, а «датаРождения» имеет тип данных в виде даты. У одного из объектов класса «Человек» атрибут «имя» имеет значение «Андрей», а «датаРождения» имеет значение «21 октября 1983 года». У другого объекта того же класса атрибут «имя» имеет значение «Мария», а атрибут «датаРождения» имеет значение «10 февраля 1970 года».

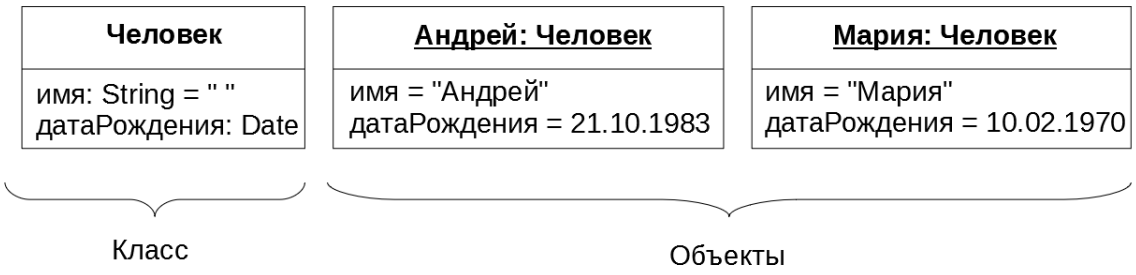


Рисунок 4.2 – Отображение атрибутов в нотации UML

Согласно системе обозначений UML, атрибуты указываются во втором (сверху) отделе прямоугольника, обозначающего класс. После каждого атрибута могут быть указаны необязательные сведения о нем (например, тип и значение по умолчанию). Перед значением по умолчанию ставится знак равенства. Название атрибута указывается обычным шрифтом (не полужирным) и первая буква не делается заглавной.

Иногда среда реализации может требовать наличия у объекта уникального идентификатора. Идентификаторы всегда неявным образом присутствуют в модели классов. Их не следует указывать явно. Большинство объектно-ориентированных языков генерируют идентификаторы для ссылок на объекты автоматически. Так же легко определить идентификаторы и для баз данных. Идентификаторы являются артефактом вычислительной системы и не имеют внутреннего смысла.

Не следует путать внутренние идентификаторы с атрибутами. Внутренние идентификаторы вводятся исключительно для удобства реализации и не имеют никакого смысла в контексте приложения. Такие атрибуты, как индивидуальный номер налогоплательщика, серийный номер и телефонный номер, не являются внутренними идентификаторами, поскольку они имеют значение в реальном мире. Их можно назвать полноправными атрибутами.

*Операция* – это функция или процедура, которая может быть применена к объектам класса. Операциями класса «Компания» могут быть «нанять», «уволить», «выплатитьДивиденды». Все объекты одного класса имеют общий список операций. Каждая операция в качестве неявного аргумента принимает свой целевой объект. Поведение операции зависит от класса целевого объекта. Объект всегда знает свой собственный класс, а потому он всегда знает и правильную реализацию операции.

Одна и та же операция может быть применена к разным классам. Такая операция называется полиморфной: в разных классах она может принимать разные формы. Методом называется реализация операции в конкретном классе. например, класс «Файл» может иметь операцию «печать». Печать текстовых файлов, двоичных файлов и цифровых изображений может осуществляться разными методами. Все эти методы с логической точки зрения выполняют одно и то же действие: печать файла. Поэтому они

и являются реализациями одной и той же операции «печать». При этом каждый метод может быть реализован своим собственным кодом.

У операции могут быть и другие аргументы, кроме целевого объекта. Эти аргументы могут быть как значениями, так и другими объектами. Выбор метода зависит только от класса целевого объекта, но не от классов аргументов, которые являются объектами, сколько бы их ни было. В некоторых объектно-ориентированных языках выбор метода может определяться произвольным числом аргументов, но такая универсальность значительно усложняет семантику модели.

Если операция реализована несколькими методами в разных классах, очень важно, чтобы у всех методов была одна и та же сигнатура – количество и типы аргументов, а также тип возвращаемого значения. Поведение всех методов операции должно быть согласованно. Лучше всего избегать использования одинаковых названий для операций, отличающихся друг от друга с семантической точки зрения, даже если они применяются к разным множествам классов. Например, неправильно было бы использовать название «инверсия» для операций инвертирования матрицы и переворота геометрической фигуры. В большом проекте может потребоваться использование пространств имен для предотвращения возможных проблем.

На рисунке 4.3 изображен класс «ГеометрическаяФигура». В нем есть операции «перемещение», «выделение» и «поворот». Операция «перемещение» имеет один аргумент «дельта» типа «Вектор», операция «выделение» имеет аргумент «т» типа «Точка» и возвращает значение логического типа. Операция «поворот» имеет аргумент «угол», который относится к типу чисел с плавающей точкой и имеет значение по умолчанию 0.0.

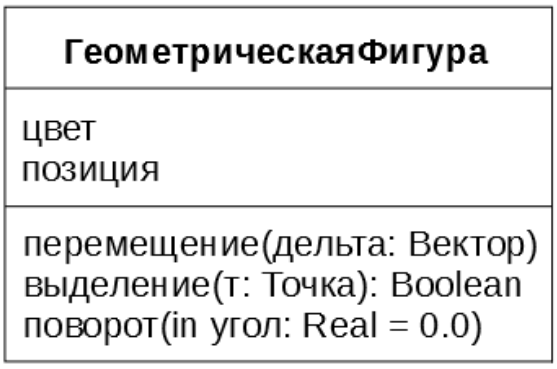


Рисунок 4.3 – Отображение операций в нотации UML

Система обозначение UML предписывает перечислять операции в третьем отделе прямоугольника класса. Название операции мы указываем обычным шрифтом и выравниваем по левому краю. Первую букву названия мы не делаем заглавной. После названия операции могут быть указаны необязательные дополнительные сведения, такие как список аргументов и тип возвращаемого результата. Список аргументов указывается в круглых скобках. Аргументы отделяются друг от друга запятыми. Перед типом возвращаемого результата ставится двоеточие. Пустой список аргументов в круглых скобках явно показывает, что данная функция не принимает никаких аргументов. Если же списка просто нет, никаких выводов делать нельзя. Мы не указываем операции для отдельных объектов, поскольку у всех объектов одного класса операции одинаковые.

Каждый аргумент может быть охарактеризован направлением, названием, типом и значением по умолчанию. По направлению аргумент может быть входным (in), выходным (out) или изменяемым (inout). Перед типом аргумента ставится двоеточие. Перед значением по умолчанию ставится знак равенства. Это значение используется в том случае, если при вызове операции значение аргумента не было указано явно.

Отделы со списком атрибутов и операций являются необязательными элементами системы обозначений. Отсутствие отдела атрибутов говорит о том, что в данном представлении атрибуты не указаны. Отсутствие отдела операций также говорит о том, что в данном представлении не указаны операции. Напротив, наличие пустого отдела говорит о том, что атрибуты или операции отсутствуют.

### 4.3.2 Концепции связи и ассоциации

*Связь* – это физическое или концептуальное соединение между объектами. Например, *Андрей работает на компанию «Симплекс»*. В большинстве случаев связь соединяет ровно два объекта, но бывают связи, соединяющие большее количество объектов. С математической точки зрения связь является кортежем, то есть списком объектов. Связь — это экземпляр ассоциаций.

*Ассоциация* – это описание группы связей, обладающих общей структурой и общей семантикой. Например, *человек может работать на какую-либо компанию.* Связи, являющиеся экземплярами некоторой ассоциации, соединяют объекты тех классов, которые соединены между собой этой ассоциацией. Ассоциация описывает множество потенциальных связей точно так же, как класс описывает множество потенциальных объектов. Связи и ассоциации обычно присутствуют в постановке задачи в виде глаголов.

На рис. 4.4 и 4.5 приведен пример диаграммы классов (рис. 4.4) и диаграмма объектов (рис. 4.5).

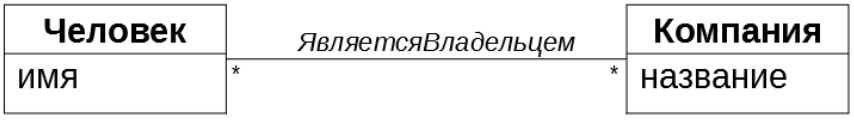


Рисунок 4.4 – Отображение ассоциаций в нотации UML


Рисунок 4.5 – Отображение связей в нотации UML

Диаграмма классов показывает, что человек может быть владельцем акций неопределенного количества компаний (от нуля и более). С другой стороны, владельцами акций одной компании могут быть несколько человек. На диаграмме объектов показаны примеры. Андрей, Мария и Евгения являются владельцами акций компании «Тент». Евгения и Анна являются владельцами акций компании «Сент». Антон не купил никаких акций, а потому и связи для него нет.

Система обозначений UML предписывает изображать связь как линию между двумя объектами. Линия может состоять из нескольких прямолинейных сегментов. Ассоциация соединяет между собой классы и тоже обозначается линией (которая тоже может иметь несколько прямолинейных сегментов). По нашему соглашению названия связей и ассоциаций выделяются курсивом, а сегменты линий привязываются к прямоугольной сетке. Удобно, по возможности, расставлять классы так, чтобы ассоциация читалась слева направо.

Название ассоциации указывать не обязательно, если в модели не возникает двусмысленности. Неоднозначность появляется в тех случаях, когда между одними и теми же классами существуют несколько ассоциаций, например, *человек работает на компанию* и *человек является владельцем акций компании.* В этом случае необходимо использовать имена ассоциаций или имена полюсов ассоциаций.

Ассоциации по своей сути являются двусторонними. Название бинарной ассоциации обычно читается в конкретном направлении, но сама ассоциация может быть прослежена в любом направлении. Например, ассоциация *работает на* соединяет

человека и компанию. Обратная ассоциация могла бы называться *оплачивает услуги,* и она соединяла бы компанию и человека. В реальности оба направления ассоциации имеют одинаково важное значение и относятся к одной и той же ассоциации. Только названия ассоциаций задают им определенные направления.

Разработчики часто реализуют ассоциации в виде ссылок из одного объекта на другой. Ссылка – это атрибут объекта, ссылающийся на другой объект. Например, структура данных класса «Человек» может содержать атрибут «работодатель», ссылающийся на объект класса «Компания», а объект класса «Компания» может содержать атрибут «сотрудники», ссылающийся на множество объектов класса «Человек».

В объектно-ориентированном подходе подчеркивается важность инкапсуляции, которая состоит в скрытии деталей реализации внутри класса. Ассоциации особенно важны потому, что они нарушают инкапсуляцию. Ассоциации не могут быть скрыты внутри класса, поскольку они соединяют разные классы. Ассоциации должны считаться равноправными по отношению к классам, в противном случае в программах будут содержаться скрытые предположения и зависимости. Такие программы трудно расширять, а классы, из которых они состоят, сложно использовать повторно.

*Кратность* – это количество экземпляров одного класса, которые могут быть связаны с одним экземпляром другого класса через одну ассоциацию. Кратность ограничивает количество связанных между собой объектов. Чаще всего рассматриваются два значения кратности: «один» и «много», но общем случае кратность может быть потенциально бесконечным подмножеством неотрицательных целых чисел. На диаграммах UML кратность указывается явно около конца линии, которой обозначается ассоциация. Значение кратности указывается в виде диапазона, например, «1» (равно один), «1..*» (один и более) или «3..5» (от трех до пяти включительно). Специальный символ «*» обозначает слово «много»: нуль и более.

Не следует путать кратность с количеством элементов. Кратность – это ограничение на размер совокупности, а количество элементов – это число элементов, которые фактически входят в совокупность. Следовательно, кратность ограничивает количество элементов.

Кратность зависит от предположений и от определенных разработчиком границ задачи. Нечеткие требования часто затрудняют определение кратности. Не следует сильно беспокоиться о правильных значениях кратности на начальных этапах разработки системы. Сначала следует определить классы и ассоциации, а затем указывать кратность. Если кратность не указана на диаграмме, она считается неопределенной.

Кратность неявно подразумевает наличие полюсов ассоциации. Например, ассоциация один-ко-многим имеет два полюса, у одного из которых указана кратность «один», а у другого – «много». Концепция полюса ассоциации – одна из важнейших в UML. Полюс ассоциации может иметь не только кратность, но и свое собственное имя.

Имена полюсов ассоциаций часто присутствуют в описаниях задач в виде существительных. Имя полюса указывается около конца ассоциации. На рисунке 4.6 «Человек» и «Компания» участвуют в ассоциации *РаботаетНа*. Человек по отношению к компании является сотрудником, а компания по отношению к человеку – работодателем. Использование имен полюсов ассоциаций не является обязательным, но чаще всего оказывается проще указать имена полюсов вместо имен ассоциаций или, по крайней мере, вместе с ними.


Рисунок 4.6 – Отображение имен полюсов ассоциации в нотации UML

Имена полюсов ассоциаций особенно удобны для прослеживания ассоциаций, потому что каждый из них может рассматриваться как псевдоатрибут. Каждый полюс бинарной ассоциации ссылается на объект или множество объектов, связанных с исходным объектом. С точки зрения исходного объекта, прослеживание ассоциации – это операция, которая возвращает связанные с ним объекты. Имя полюса ассоциации – это средство для прослеживания ассоциации без явного ее указания.

Имена полюсов позволяют унифицировать несколько ссылок на один и тот же класс. При построении диаграмм классов следует корректно использовать имена полюсов ассоциаций и не вводить отдельный класс для каждой ссылки. На рис. 4.7 человек с ребенком может быть представлен двумя экземплярами: один для родителя и один для ребенка. В корректной модели экземпляр класса «Человек» принимает участие в двух и более связях: дважды в качестве родителя и произвольное количество раз в качестве ребенка.

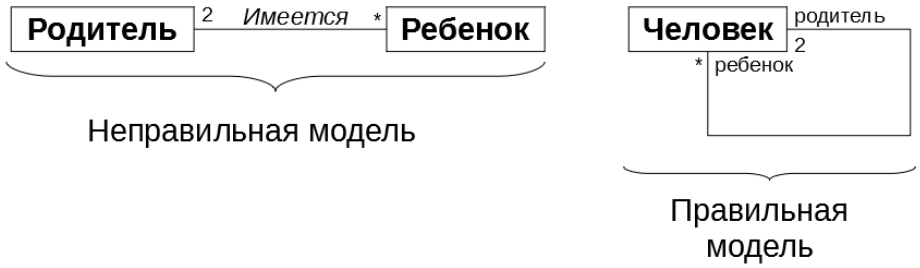


Рисунок 4.7 – Моделирование ссылок на один и тот же класс

Поскольку имена полюсов ассоциации позволяют отличать объекты друг от друга, все имена на дальнем полюсе ассоциации, прикрепленной к некоторому классу, должны быть уникальными. Хотя имя ставится около целевого объекта ассоциации, фактически оно является псевдоатрибутом исходного класса, а потому должно быть уникально внутри него. По той же причине имя полюса ассоциации не должно совпадать с именем какого-либо атрибута исходного класса.

Подобно тому, как объекты класса могут быть описаны при помощи атрибутов, связи ассоциации также могут быть описаны атрибутами. UML позволяет представлять информацию такого характера при помощи классов ассоциаций.

*Класс ассоциаций* – это ассоциация, которая одновременно является классом. Подобно связям ассоциаций, экземпляры класса ассоциаций обладают индивидуальностью, связанной с теми объектами, между которыми они проводятся. Подобно обычным классам, классы ассоциаций могут иметь атрибуты и операции и участвовать в ассоциациях. Классы ассоциаций присутствуют в формулировке задачи в виде наречий или получаются абстрагированием известных значений.

На рис. 4.8 «разрешениеДоступа» является атрибутом «Доступно». В UML класс ассоциации обозначается прямоугольником, прикрепленным к ассоциации пунктирной линией.

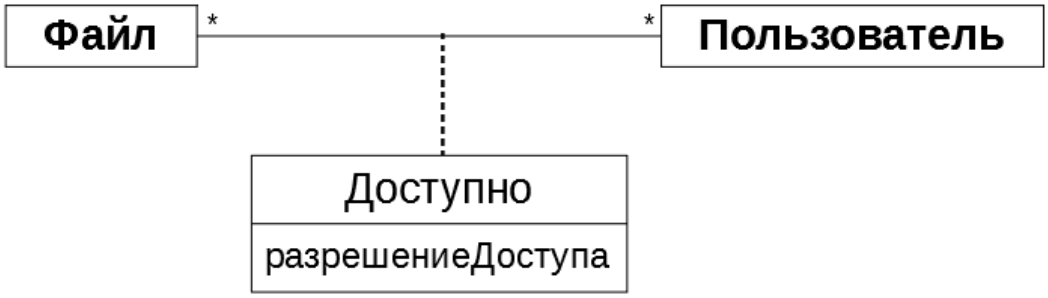


Рисунок 4.8 – Атрибуты связи ассоциаций

Основанием для введения классов ассоциаций послужила возможность создания ассоциаций многие-ко-многим. Атрибуты таких ассоциаций без всяких сомнений являются принадлежностью связей и не могут быть приписаны ни к одному из объектов-участников. На рис. 4.8 атрибут «разрешениеДоступа» относится к файлу и пользователю одновременно и не может быть прикреплен только к одному из них без потери информации.

*Квалифицированной* называется ассоциация, у которой имеется специальный атрибут (квалификатор), используемый для того, чтобы отличать друг от друга объекты, находящиеся на полюсе ассоциации с кратностью «много». Квалификаторы могут быть определены для ассоциаций типа один-ко-многим и многие-ко-многим. Квалификатор позволяет выбрать отдельный объект из множества целевых объектов, уменьшая таким образом эффективную кратность до значения «один». Квалифицированная ассоциация с целевой кратностью «один» или «не более одного» образует четкий маршрут для поиска целевого объекта по исходному.

На рисунке 4.9 демонстрируется наиболее типичный пример использования квалификатора для ассоциаций с кратностью один-ко-многим. В банке имеются множество счетов. Счет принадлежит одному единственному банку. В контексте банка уникальный счет определяется своим номером. «Банк» и «Счет» – это классы, а «номерСчета» – квалификатор. Квалификация уменьшает эффективную кратность ассоциации до единицы.

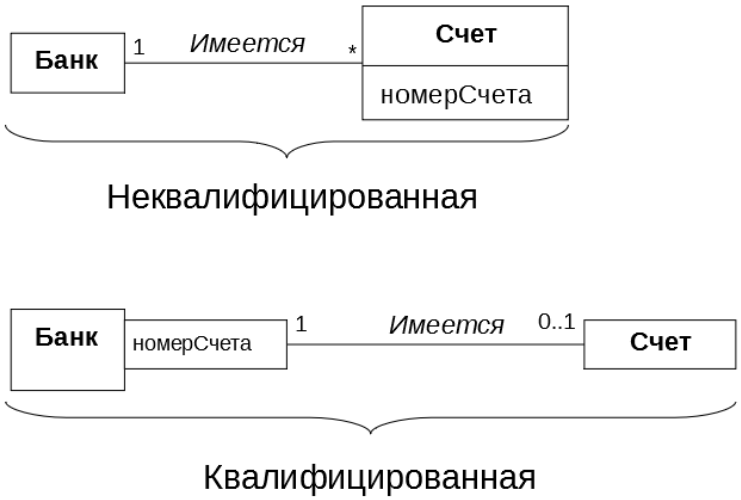


Рисунок 4.9 – Квалифицированная ассоциация

Обе модели (квалифицированная и неквалифицированная) вполне корректны, но модель с квалифицированной ассоциацией сообщает дополнительную информацию. В квалифицированной модели добавляется ограничение на кратность: сочетание банка и номера счета дает не более одного счета. Квалифицированная модель также передает значение номера счета для прослеживания модели, что отражается в методах. Сначала следует найти банк, а затем указать номер счета, и в результате вы получите нужный счет.

Для обозначения квалификатора используется небольшой прямоугольник, который пристыковывается к исходному классу около конца линии, обозначающей ассоциацию. Квалификатор может быть пристыкован к любой стороне прямоугольника исходного класса. Исходный класс вместе с квалификатором определяют целевой класс.

### 4.3.3 Обобщение и наследование

*Обобщение* – это отношение между классом (суперклассом) и одной или несколькими его вариациями (подклассами). Обобщение объединяет классы по их общим свойствам, благодаря чему обеспечивается структурирование описания объектов. Суперкласс характеризуется общими атрибутами, операциями и ассоциациями. Подклассы добавляют к ним свои собственные атрибуты, операции и ассоциации. Говорят, что подкласс наследует составляющие суперкласса. Обобщение иногда называется отношением типа «является», поскольку каждый экземпляр подкласса одновременно *является* экземпляром суперкласса.

Простое обобщение упорядочивает классы в рамках некоторой иерархии. В этом случае каждый подкласс имеет одного непосредственного предка (его суперкласс). Уровней обобщения может быть много. Примеры обобщений приведены на рис. 4.10.

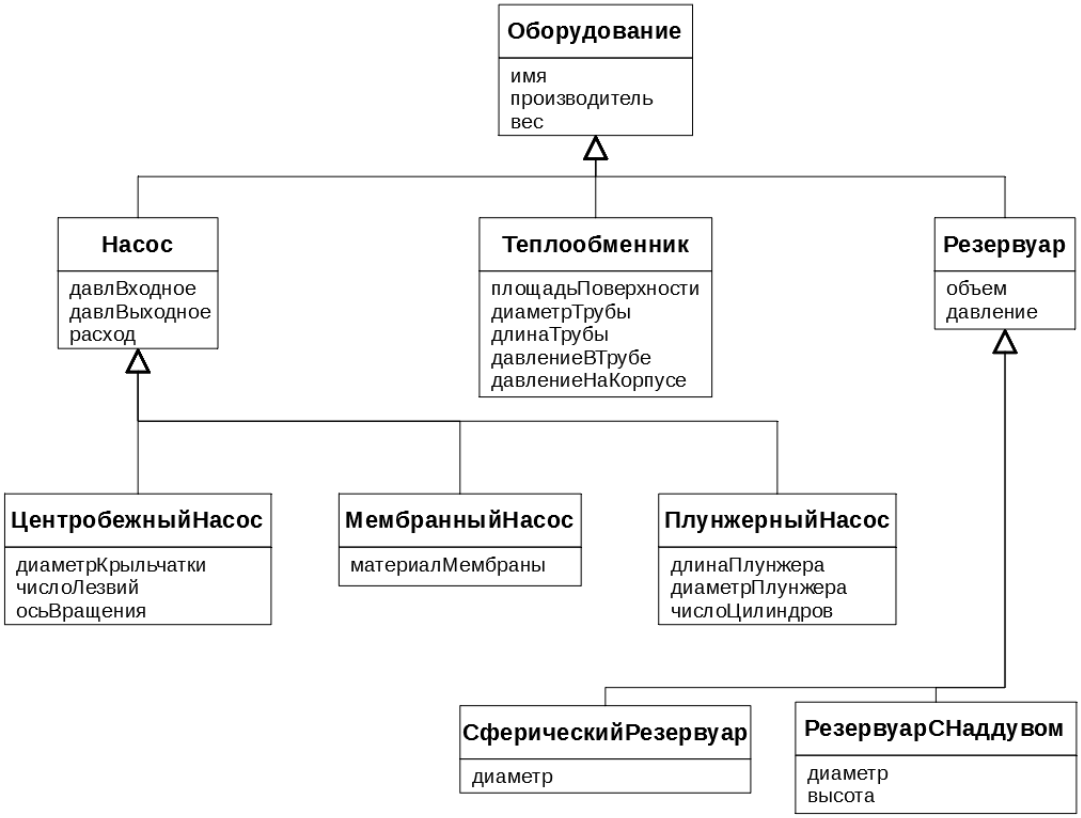


Рисунок 4.10 – Многоуровневая иерархия наследования с экземплярами

Обобщение обозначается большой не закрашенной стрелкой. Стрелка указывает на суперкласс. Суперкласс можно соединять с каждым из его подклассов непосредственно, но предпочтительнее будет группировать обобщения в дерево. Треугольник можно повернуть и расположить его с любой стороны от суперкласса, но, по возможности, следует изображать суперкласс сверху, а его подклассы снизу.

Обобщение транзитивно и действует через произвольное количество уровней иерархии. Термины «предок» и «потомок» используются для описания классов, находящихся далеко друг от друга по уровням, но связанных отношением обобщения. Экземпляр подкласса одновременно является экземпляром всех его предков. Экземпляр обладает значениями всех атрибутов всех классов-предков. Экземпляр может вызывать любую операцию, указанную у любого из его предков. Подклассы не только наследуют все составляющие своих предков, но и добавляют к ним свои собственные составляющие.

Обобщение служит трем основным целям. Первая цель – обеспечение поддержи полиморфизма. Операция может быть вызвана на уровне суперкласса, а компилятор объектно-ориентированного языка автоматически разрешит вызов в метод, соответствующий классу вызывающего объекта. Полиморфизм увеличивает гибкость программного обеспечения: вы добавляете новый подкласс и автоматически наследуете поведение суперкласса. Более того, новый подкласс не нарушает работу существующего кода.

Вторая цель обобщения состоит в структурировании описаний объектов. Используя обобщение, вы делаете концептуальное утверждение, образуя таксономию и

упорядочивая объекты на основании их сходств и различий. Такой подход гораздо более основателен, нежели моделирование каждого класса в изоляции от всех остальных.

Третья цель состоит в обеспечении повторного использования кода: вы можете наследовать код в рамках одного приложения, а также из существующих приложений (библиотек классов). Повторное использование повышает производительность по сравнению с многократным переписыванием кода «с нуля». Обобщение позволяет корректировать код для получения именно такого поведения, которое требуется в данном приложении.

Подкласс может подменять или перекрывать составляющую суперкласса, определяя составляющую с тем же именем внутри себя. Подмена составляющих может потребоваться по нескольким причинам: для спецификации поведения, зависящего от подкласса; для уточнения спецификации составляющей; для повышения производительности.

Подменять можно методы и значения по умолчанию для атрибутов. Никогда не следует подменять сигнатуру составляющей. Подмена должна сохранять тип атрибута, количество и тип аргументов операции, а также тип возвращаемого операцией значения. Уточнение типа атрибута или аргумента операции (то есть указание в качестве типа подтипа исходного типа) является формой ограничения и должно использоваться с осторожностью.

## 4.4 Моделирование состояний

### 4.4.1 События

*Событие* – это происшествие, случившееся в определенный момент времени, например, «нажатие левой кнопки мыши». Часто события соответствуют глаголам в прошедшем времени (питание было включено, будильник был установлен) или выполнению некоторого условия (опустошился лоток для бумаги) в описании задачи. По определению, событие происходит мгновенно, по крайней мере, во временном масштабе системы. Событие рассматривается как атомарное и скоротечное происшествие. Неявным атрибутом события является момент его осуществления. Продолжительные изменения, осуществляющиеся в течение некоторого промежутка времени, хорошо описываются с помощью концепции состояния.

Одно событие может логически предшествовать другому или следовать за ним. События могут быть и несвязанными друг с другом. Несвязанные события считаются параллельными. Они никак не влияют друг на друга. Если временная задержка при передаче информации между двумя точками превышает временной интервал между событиями, эти события обязаны быть параллельными, поскольку они никак не могут повлиять друг на друга. При моделировании системы не следует задавать относительный порядок параллельных событий, потому что на практике они могут происходить в любом порядке.

К событиям относятся не только нормальные происшествия, но и ошибочные ситуации. Ошибочная ситуация ничем не отличается от любого другого события. Только в нашей интерпретации она становится «ошибкой».

Термин «событие» часто используется в нескольких смыслах. В некоторых случаях он обозначает экземпляр, а в других — класс. На практике эта двусмысленность обычно не создает проблем, так как точный смысл слова следует из контекста. При необходимости можно употреблять точные термины: «осуществление события» и

«происшествие» (экземпляр события) и «тип события» (класс). События бывают разных видов. Чаще всего встречаются события сигналов, события изменения и события времени.

*Сигнал* – это явная односторонняя передача информации от одного объекта к другому. Объект, передающий сигнал другому объекту, может рассчитывать на получение ответа, но этот ответ будет отдельным сигналом, и его отправка (или задержка) будет целиком зависеть от второго объекта.

*Событие сигнала* – это событие получения или отправки сигнала. Обычно более важным считается получение сигнала, потому что оно влияет на объект получатель. Обратите внимание на разницу между сигналом и событием сигнала: сигнал – это сообщение между объектами, а событие сигнала – это происшествие.

Каждая передача сигнала является уникальным происшествием, но мы группируем их в классы сигналов и даем каждому классу имя, подчеркивая общую структуру и поведение. Например, «вылет рейса самолета № 123 авиакомпании «Аэрофлот» из аэропорта Томска 10 октября» – это экземпляр класса сигналов «ВылетРейса» (рис. 4.11). Некоторые сигналы являются обычными происшествиями, но большинство из них характеризуются атрибутами, в которых хранятся передаваемые этими сигналами значения.

В UML сигнал обозначается ключевым словом «signal» в угловых кавычках («»), которое ставится над именем класса сигнала в верхнем разделе прямоугольника. Во втором разделе указываются атрибуты сигнала.

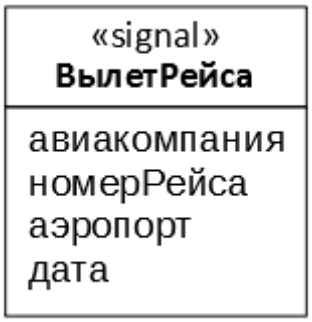


Рисунок 4.11 – Класс сигналов и его атрибуты

*Событие изменения* – это событие, вызванное выполнением логического выражения. Суть события состоит в том, что некоторое выражение постоянно проверяется, и как только его значение изменяется с «ложно» на «истинно», осуществляется событие изменения.

В UML событие изменения обозначается ключевым словом «when», за которым следует логическое выражение в круглых скобках: when(заряд аккумулятора < нижнее ограничение); when(температура в комнате > температура охлаждения).

*Событие времени* – это событие, вызванное достижением момента абсолютного времени или истечением временного интервала. В UML момент абсолютного времени обозначается ключевым словом «when», за которым следует временное выражение в круглых скобках. Временной интервал обозначается ключевым словом «after», за

которым следует выражение, результатом вычисления которого является временной интервал: when(дата = ‘1 января 2000 г.’); after(10 секунд).

### 4.4.2 Состояния

*Состояние* – это абстракция значений и связей объекта. Множества значений и связей группируются в состояние в соответствии с массовым поведением объектов. Например, состояние банка может быть «платежеспособный» или «банкрот», в зависимости от того, что больше: активы или обязательства. Состояния часто соответствуют отглагольным формам или деепричастиям («Ожидает», «Дозванивается») или выполнению некоторого условия («Включен», «НижеТочкиЗамерзания»).

На рис. 4.12 показана система обозначений состояния в UML: прямоугольник со скругленными углами, в котором ставится название состояния. Мы выделяем название состояния полужирным шрифтом, центрируем его и пишем с заглавной буквы.


Рисунок 4.12 – Состояние

Определяя состояния, мы не учитываем атрибуты, не оказывающие влияния на поведение объекта, и объединяем вместе в одно состояние все комбинации значений и связей, характеризующиеся одинаковыми откликами на события. Разумеется, каждый атрибут должен влиять на поведение, иначе он не будет иметь никакого значения, однако, достаточно часто некоторые атрибуты не влияют на последовательность управления. Их можно рассматривать просто как значения параметров состояния.

Объекты класса обладают конечным числом возможных состояний. В конкретный момент времени каждый объект может находиться ровно в одном состоянии. Объекты могут проходить через одно или несколько состояний в течение времени своего существования. В конкретный момент времени разные объекты класса могут охватывать широкий спектр состояний.

Состояние описывает отклик объекта на получаемые события. В конкретном состоянии игнорируются любые события, за исключением тех, поведение при получении которых описано явным образом. Отклик на событие может быть вызовом поведения или изменением состояния. Например, если в состоянии «Гудок» нажать кнопку с цифрой на телефоне, зуммер сбрасывается, а телефонная линия переходит в состояние «НаборНомера». Если же в состоянии «Гудок» повесить трубку, линия отключается и переходит в состояние «Свободно».

Между событиями и состояниями существует некоторая симметрия. События – это точки на линии времени, а состояния – интервалы. Состояние соответствует интервалу между двумя точками, обозначающими два полученных объектом события. Например, между снятием трубки и нажатием первой цифры, телефонная линия находится в состоянии «Зуммер». Состояние объекта зависит от предыдущих событий, которые в большинстве случаем перекрываются последующими событиями. Например, события, произошедшие до того, как трубка была повешена, не влияют на будущее

поведение. Состояние «Свободно» «забывает» о событиях, полученных до сигнала «повесить трубку».

Состояние можно характеризовать текстовым описанием. Приведем описание состояния «ЗвонокБудильника» для класса «Будильник» (рис. 4.13, табл. 4.1). Последовательность событий, которая приводит к этому состоянию, состоит из: 1) установки времени звонка будильника; 2) произвольных действий, не приводящих к его сбрасыванию, и 3) наступления заданного момента времени. Условие состояния выражается в терминах параметров, таких как текущее и целевое время. Звонок прекращается после 20 секунд. Таблица событий и откликов показывает реакцию на события «текущее время» и «нажатие кнопки». В этой таблице указывается не только действие, но и следующее состояние. Разные описания состояния могут перекрываться.

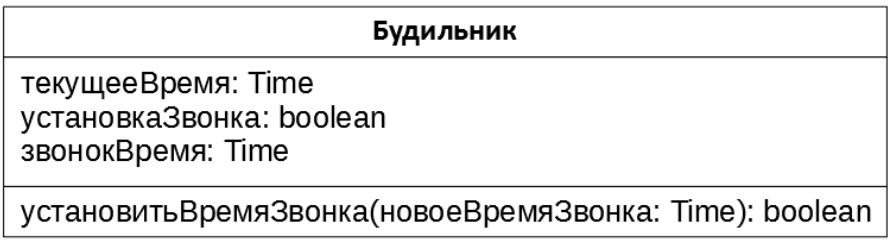


Рисунок 4.13 – Класс «Будильник»

Таблица 4.1 – Описание состояния «ЗвонокБудильника» для класса «Будильник».

| Наименование состояния | ЗвонокБудильника | |
|---|---|---|
| Описание состояния | Звучит звонок будильника по достижении ранее установленного времени звонка | |
| Идентификация состояния | 1) установкаЗвонка == true<br>2) звонокВремя != null | |
| События, приводящие к данному состоянию | 1) установитьВремяЗвонка(новоеВремяЗвонка)<br>2) любые действия, не включающие сброса атрибута «установкаЗвонка»<br>3) when(текущееВремя == звонокВремя) | |
| Условия, характеризующие данное состояние | 1) when(звонокВремя != null)<br>2) when(звонокВремя <= текущееВремя <= звонокВремя + 20 секунд)<br>3) звучит звуковой сигнал-мелодия звонка<br>4) не нажимается никакая кнопка будильника | |
| События, возможные в данном состоянии | | |
| **Событие** | **Отклик** | **Следующее состояние** |
| when(текущееВремя > звонокВремя + 20 секунд) | Прекращение звучания звукового сигнала-мелодии | «ОжиданиеВремениЗвонка» |
| Нажата любая кнопка у будильника | Прекращение звучания звукового сигнала-мелодии | «ОжиданиеВремениЗвонка» |

### 4.4.3 Переходы и условия

*Переход* – это мгновенная смена одного состояния другим. Например, когда вы отвечаете на входящий звонок, телефонная линия переходит из состояния «Звонок» в состояние «Разговор». Говорят, что переход запускается при смене исходного состояния целевым. Переход запускается, когда происходит связанное с ним событие (если только необязательное сторожевое условие не приводит к игнорированию события). Выбор целевого состояния зависит как от исходного состояния, так и от полученного события. Событие может вызвать переходы во множестве объектов. С концептуальной точки зрения эти переходы происходят одновременно.

*Сторожевое условие* – это логическое выражение, которое должно быть истинным, чтобы переход мог запуститься. Например, сигнал светофора на перекрестке может переключиться только в том случае, если на дороге имеются ожидающие этого машины. Переход со сторожевым условием запускается в тот момент, когда осуществляется соответствующее событие, но только если в этот же момент выполнено его сторожевое условие. Например, «когда будешь выходить из дома (событие), если на улице будет ниже нуля (условие) – надень перчатки (целевое состояние)». Сторожевое условие проверяется только один раз, в тот момент, когда осуществляется событие и, если условие выполняется, – происходит переход. Обратите внимание, что сторожевое условие концептуально отличается от события: условие проверяется только один раз, тогда как наличие события, по сути, проверяется непрерывно.

На рис. 4.14 показаны переходы со сторожевыми условиями для светофоров на перекрестке. Одна пара фотоэлементов контролирует полосы в направлении север-юг из которых возможен поворот налево. Другая пара контролирует полосы в направлении запад-восток из которых тоже возможен поворот налево. Если на одной из пар полос отсутствуют машины, управляющая логика светофора пропускает часть цикла, разрешающую левый поворот.

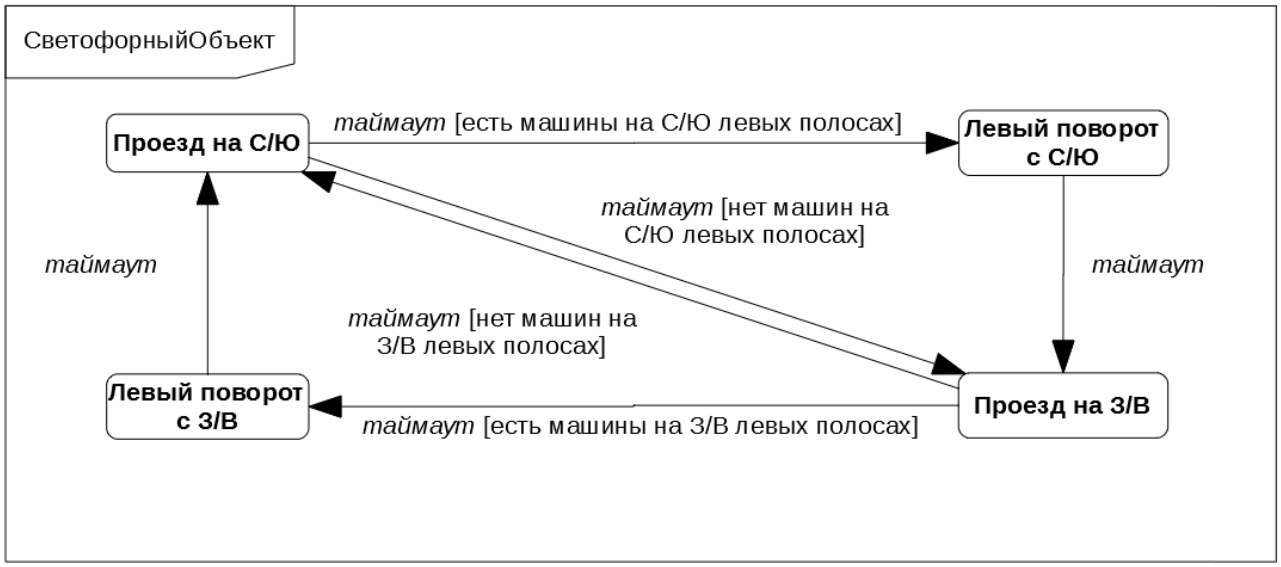


Рисунок 4.14 – Переходы со сторожевыми условиями

В UML для обозначения перехода используется линия, соединяющая исходное состояние с целевым. на одном из концов линии, указывающее на целевое состояние, ставится стрелка. Линия может состоять из нескольких сегментов. Событие может быть указано в качестве метки перехода. После события в квадратных скобках можно указать необязательное сторожевое условие. Название события выделяется курсивом, а сторожевое условие записывается в квадратных скобках.

### 4.4.4 Диаграммы состояний

*Диаграмма состояний* – это граф, узлами которого являются состояния, а направленными дугами – переходы между состояниями. Диаграмма состояний описывает последовательности состояний, вызываемые последовательностями событий. Названия состояний должны быть уникальными в рамках диаграммы. Все объекты класса следует диаграмме состояний, описывающей общее для них поведение. Диаграмма состояний может быть реализована непосредственной интерпретацией или преобразованием семантики в эквивалентный программный код.

Модель состояний состоит из множества диаграмм состояний, по одной на каждый класс, поведение которого с течением времени важно для приложения. Диаграммы состояний должны быть согласованы по интерфейсам (событиям и сторожевым условиям). Отдельные диаграммы взаимодействуют друг с другом посредством передачи событий, а также косвенно, через сторожевые условия. Некоторые события и сторожевые условия присутствуют только на одной диаграмме, тогда как другие – на нескольких.

Класс, имеющий несколько состояний, характеризуется важным поведением во времени. Если же класс обладает одним состоянием, его поведение во времени можно игнорировать. Диаграммы состояний с одним состоянием можно описать в простой форме без всякой графики, а именно в виде таблицы воздействий и откликов, в которой будут приводиться события и сторожевые условия, а также вызываемое ими поведение.

Для обозначения диаграммы состояний в UML используется прямоугольник (рис. 4.14). Название диаграммы указывается в пятиугольном теге в левом верхнем углу. Внутри прямоугольника изображаются состояния и переходы, образующие диаграмму.

Если из состояния выходит несколько переходов, первое осуществившееся событие запускает соответствующий ему переход. Если происходит событие, которому не сопоставлен ни один переход, оно просто игнорируется. Если событию соответствует несколько переходов, выбран будет только один из них, и притом случайным образом.

Диаграммы состояний могут описывать непрерывные циклы или одноразовые жизненные циклы. Диаграмма состояний светофорного объекта является непрерывным циклом. Описывая обычные режимы использования светофора, мы не интересуемся тем, каким образом цикл был запущен.

Одноразовые диаграммы состояний описывают объекты с конечным сроком существования. Такие диаграммы имеют начальное и конечное состояния. Сразу после создания объект оказывается в начальном состоянии. Вход в конечное состояние означает уничтожение объекта. На рис. 4.15 показан упрощенный жизненный цикл игры в шахматы с начальным состоянием по умолчанию (сплошной кружок) и конечным состоянием по умолчанию («бычий глаз»).

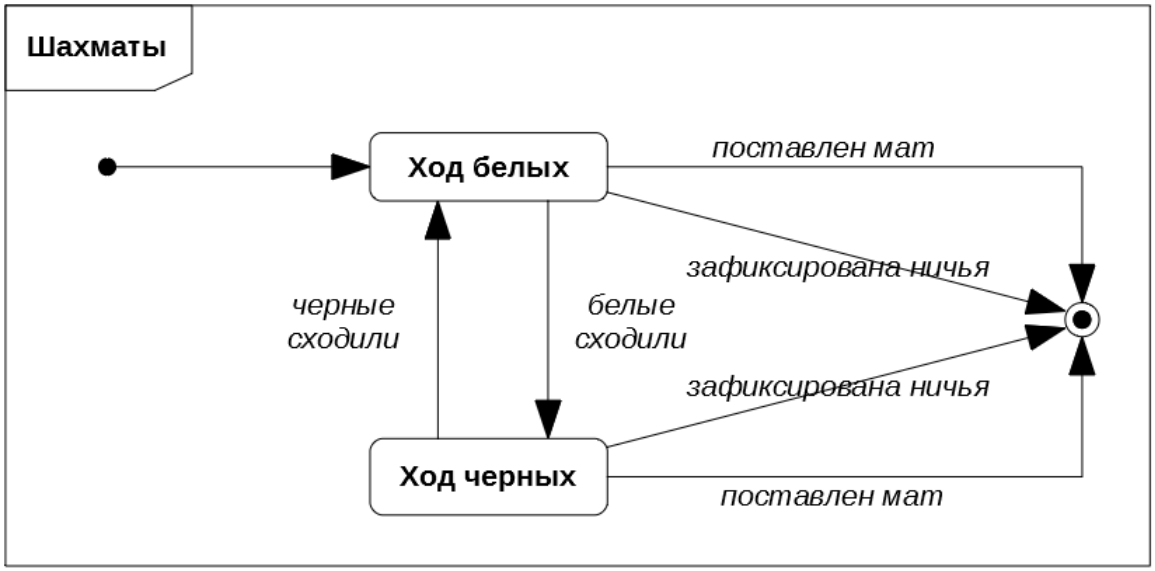


Рисунок 4.15 – Диаграмма состояний игры в шахматы

Начальное и конечное состояния можно обозначать точками входа и выхода. На рис. 4.16 точка входа «Начало» соединена с первым ходом белых, и игра заканчивается одним из трех возможных состояний. Точки входа (пустые кружки) и выхода (кружки с символом «X») ставятся на периметре диаграммы состояний и могут иметь имена.

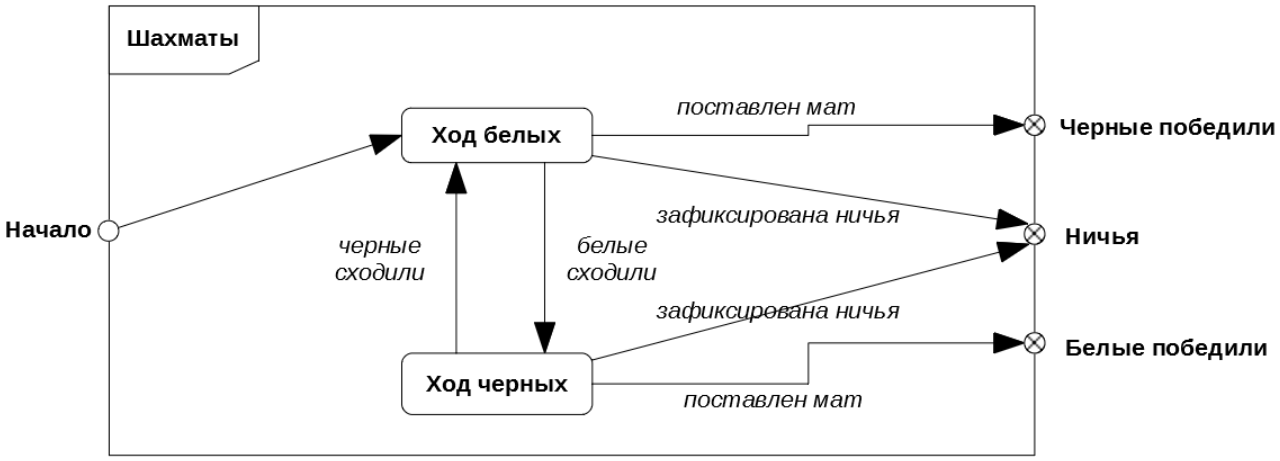


Рисунок 4.16 – Диаграмма состояний игры в шахматы с точками входа и выхода

### 4.4.5 Поведение на диаграммах состояний

Диаграммы состояний были бы не слишком полезны, если бы они описывали только события. Полное описание объекта должно указывать, что именно делает объект в ответ на события.

*Действие* – это ссылка на поведение, выполняемое в ответ на произошедшее событие. *Деятельность* – это фактическое поведение, которое может вызываться любым количеством действий. Например, деятельность «разъединитьЛинию» может выполняться при переходе, при входе в состояние или при выходе из него, а также при наступлении какого-либо иного события в состоянии.

Деятельность может описывать внутренние управляющие операции, например, установку атрибутов или порождение других событий. Эта деятельность не имеет аналогов в реальном мире и предназначена для структурирования управления при реализации. Например, программа может увеличивать внутренний счетчик на единицу каждый раз при осуществлении какого-либо события.

Деятельность обозначается косой чертой (/), после которой ставится название или описание деятельности. Деятельность указывается после вызывающего ее события. На рис. 4.17 показана диаграмма состояний для всплывающего меню рабочей станции. При нажатии правой кнопки мыши меню отображается на экране. Когда пользователь отпускает эту кнопку, меню исчезает. Пока меню отображается на экране, в нем подсвечивается один элемент, над которым в данным момент находится указатель мыши.

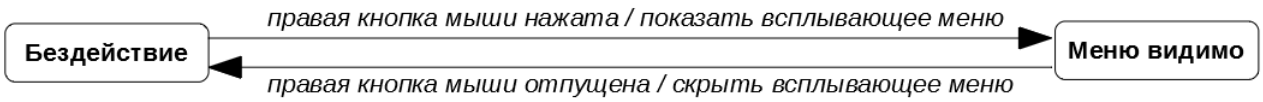


Рисунок 4.17 – Деятельность для всплывающего меню

*Текущей* называется деятельность, занимающая некоторый промежуток времени. По определению, такая деятельность может выполняться только в некотором состоянии и не может прикрепляться к переходу. Например, индикатор аварии у принтера может мигать в состоянии «Замятие бумаги» (см. рис. 4.18). Текущая деятельность включает непрерывные операции, такие как отображение картинки на телеэкране, а также последовательные операции, завершающиеся по прошествии некоторого промежутка времени (например, закрытие клапана).

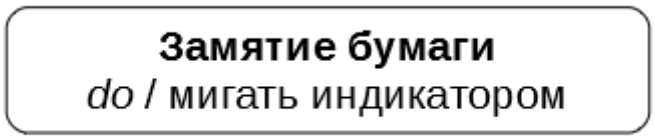


Рисунок 4.18 – Текущая деятельность при замятии бумаги у принтера

Для обозначения текущей деятельности используется ключевое слово «do» и символ косой черты. Текущая деятельность может быть прервана событием, полученным в процессе выполнения этой деятельности. Это событие может вызвать переход из состояния, в котором осуществляется текущая деятельность, но может и не

вызвать такого перехода. Например, робот, перемещающий деталь, может столкнуться с сопротивлением, что приведет к остановке его движения.

Деятельность может быть прикреплена не только к переходу, но и ко входу в состояние или к выходу из него. Никаких отличий в возможностях двух систем обозначений нет, однако, часто при всех переходах в одно и то же состояние выполняется одинаковая деятельность, которую, в таком случае, удобнее привязать к самому событию.

Часто единственным назначением состояния является последовательное выполнение некоторой деятельности. Как только деятельность завершается, запускается переход в следующее состояние. Стрелка без названия события обозначает автоматический переход, который запускается, как только завершается деятельность, связанная с исходным состоянием. Такой переход называется переходом по завершении, потому что он переключается завершением деятельности в исходном состоянии.

Сторожевое условие проверяется только один раз, в тот момент, когда осуществляется событие. Если переход имеет переход по завершении, а его сторожевое условие не выполнено, состояние остается активным и может привести к сбою управления: событие завершения деятельности не может произойти второй раз. Если состояние имеет переходы по завершении, сторожевые условия должны охватывать весь спектр возможных событий. Специальное условие «else» позволяет задать ситуацию, когда все остальные условия оказываются ложными. Не используйте сторожевые условия на переходы по завершении для ожидания изменения значения. Лучше ожидать события изменения.

## 4.5 Моделирование взаимодействий

### 4.5.1 Модели вариантов использования

*Действующее лицо* – это непосредственный внешний пользователь системы. Это объект или множество объектов, непосредственно взаимодействующих с системой, но не являющихся ее частью. Каждое действующее лицо является обобщением группы объектов, ведущих себя определенным образом по отношению к системе. Например, клиент и ремонтник являются разными действующими лицами по отношению к торговому автомату. Действующими лицами по отношению к системе бюро путешествий могут быть путешественники, агенты и авиакомпании. По отношению к компьютерной базе данных ими могут быть пользователи и администраторы. Действующими лицами могут быть люди, устройства и другие системы – все, что взаимодействует с интересующей нас системой непосредственно.

Объект может быть связан с несколькими действующими лицами, если его поведение обладает разными гранями. Например, объекты Мария и Андрей могут быть клиентами торгового автомата. Андрей может одновременно являться и ремонтником, обслуживающим этот автомат.

Действующее лицо должно иметь одну четко определенную цель. Объекты и классы обычно сочетают в себе несколько назначений (целей). Действующее же лицо представляет лишь одну грань объекта в его взаимодействии с системой. Одно и то же лицо может представлять объекты разных классов, ведущие себя одинаково по отношению к системе. Например, люди, так или иначе взаимодействующие с торговым автоматом, могут быть разделены на две основные категории: клиенты и ремонтники.

Каждое действующее лицо обладает согласованным набором возможностей тех объектов, которые оно представляет.

Моделирование действующих лиц помогает определить границы системы, то есть идентифицировать объекты, находящиеся внутри системы, и объекты, лежащие на ее границе. Действующее лицо непосредственно взаимодействует с системой. Объекты, участвующие только в косвенных взаимодействиях с системой, не являются действующими лицами и потому не должны включаться в модель системы. Любое косвенное взаимодействие должно осуществляться посредством действующих лиц. Например, диспетчер мастеров по ремонту торговых автоматов не является действующим лицом по отношению к этим автоматам. Непосредственно с автоматами работают только мастера. Если нужно построить модель косвенного взаимодействия внешних объектов, нужно строить модель среды как большой системы, включающей в себя исходную систему. Например, можно построить модель ремонтной службы, включающей в себя (в качестве действующих лиц) диспетчеров, мастеров и торговые автоматы, но эта модель уже не будет моделью торговых автоматов.

Различные взаимодействия действующих лиц с системой группируются в варианты использования. ***Вариант использования*** – это связный элемент функциональности, представляемый системой при взаимодействии с действующими лицами. Например, лицо «Клиент» может *купить напиток* в торговом автомате. Клиент кидает монеты в автомат, выбирает продукт и забирает свой напиток. «Ремонтник» может *провести техническое обслуживание* автомата.

Примеры вариантов использования для торгового автомата:

1) «Купить напиток». Торговый автомат выдает напиток после того, как клиент выбирает нужный пункт и платит за него.

2) «Провести плановый ремонт». Ремонтник выполняет плановое техобслуживание автомата, необходимое для обеспечения его безотказной работы.

3) «Провести техобслуживание». Ремонтник выполняет незапланированное обслуживание автомата при его выходе из строя.

4) «Загрузить продукты». Обслуживающий персонал загружает продукты в торговый автомат для пополнения запасов продаваемых напитков.

В каждом варианте использования участвуют одно или несколько действующих лиц и система. В варианте использования «купить напиток» участвует лицо «Клиент», а в варианте использования «провести плановый ремонт» участвует лицо «Работник». Вариант использования подразумевает обмен последовательностью сообщений между системой и действующими лицами. Например, в варианте использования «купить напиток» клиент сначала вставляет монету, а торговый автомат отображает текущий аванс. Эта процедура может быть повторена несколько раз. Затем клиент нажимает кнопку выбора товара, а автомат выдает заказанный товар и сдачу, если таковая имеется.

Некоторые варианты использования характеризуются фиксированной последовательностью сообщений. Однако чаще последовательность сообщений может варьироваться в определенных пределах. Например, клиент может кинуть в автомат некоторое количество монет (вариант использования «купить напиток»). В зависимости от количества и номинала монет, а также от выбранного товара автомат может либо вернуть сдачу, либо нет. Вариации последовательностей можно показать, продемонстрировав несколько примеров различающихся между собой последовательностей поведения. Обычно сначала определяется основная последовательность, а затем – необязательные последовательности, повторения и другие вариации.

Частью варианта использования являются и сбойные ситуации. Например, если клиент выбирает напиток, который уже кончился, торговый автомат отображает соответствующее сообщение. Кроме того, клиент может отменить транзакцию покупки. Например, он может нажать кнопку возврата монет в любой до того, как его выбор будет принят. В этом случае автомат возвращает монеты клиенты, и последовательность поведения для варианта использования завершается. С точки зрения пользователя некоторые виды поведения могут считаться ошибками. Однако проектировщик должен учитывать все возможные последовательности. С точки зрения системы ошибки пользователя или ресурсов являются разновидностями поведения, которые устойчивая система должна быть способна обработать.

Вариант использования объединяет все поведение, имеющее отношение к элементу функциональности системы: нормальное поведение, вариации нормального поведения, исключительные ситуации, сбойные ситуации и отмены запросов. Объединение нормального и аномального поведения в одном варианте использования помогает убедится, что все последовательности взаимодействия рассматриваются вместе. В полной модели варианты использования образуют разбиение функциональности системы. Все варианты должны находиться на сравнимом уровне абстракции.

Пример описания варианта использования:

– Вариант использования: Покупка напитка.

– Краткое описание: Торговый автомат выдает напиток после того, как клиент выбирает нужный вариант и платит за него.

– Действующие лица: Клиент.

– Исходные условия: Автомат ожидает опускания монеты.

– Описание: Автомат изначально находится в состоянии ожидания и выводит на дисплей сообщение «Опустите монеты». Клиент опускает монеты в щель автомата. Автомат выводит на дисплей принятую от клиента сумму и включает подсветку кнопок с названиями напитка. Автомат выдает соответствующий напиток и выдает сдачу, если напиток стоит меньше, чем заплатил клиент.

– Исключения:

1) Отмена: Если клиент нажмет кнопку отмены до того, как произведет выбор напитка, автомат вернет клиенту деньги и перейдет в состояние ожидания.

2) Напиток отсутствует: Если клиент выбирает напиток, который в данный момент отсутствует в автомате, то выводится сообщение «Напиток отсутствует». Автомат готов к приему монет и выбору напитка клиентом.

3) Недостаточно денег: Если клиент выбирает напиток, который стоит больше, чем он заплатил, то выводится сообщение «Необходимо доплатить xx.xx рублей для покупки этого напитка». Автомат продолжает принимать монеты и готов к выбору напитка клиентом.

4) Нет сдачи: Если клиент вставил достаточное количество монет для покупки напитка, но в автомате нет денег для корректной выдачи сдачи, то выводится сообщение «Невозможно выдать сдачу». Автомат продолжает принимать монеты и готов к выбору напитка клиентом.

– Постусловия: Автомат готов к приему монет.

Любая система обладает своим множеством вариантов использования и множеством действующих лиц. Каждый вариант использования описывает элемент предоставляемой системой функциональности. Множество вариантов использования описывает всю функциональность системы на некотором уровне абстракции. Каждое

действующее лицо представляет собой один вид объектов, для которых система может выполнять некоторое поведение. Множество действующих лиц описывает полное множество объектов, которые могут быть обслужены системой. Объекты аккумулируют поведение всех систем, с которыми они взаимодействуют в качестве действующих лиц.

Язык UML предусматривает систему графических обозначений для вариантов использования (см. рис. 4.19). Варианты использования системы заключаются в прямоугольник, снаружи которого изображаются действующие лица. Название системы может быть указано около одного из краев прямоугольника. Вариант использования обозначается эллипсом, внутри которого указывается его название. Значок «человечек» обозначает действующее лицо. Его имя ставится рядом со значком или под ним. Действующие лица соединяются с вариантами использования сплошными линиями.

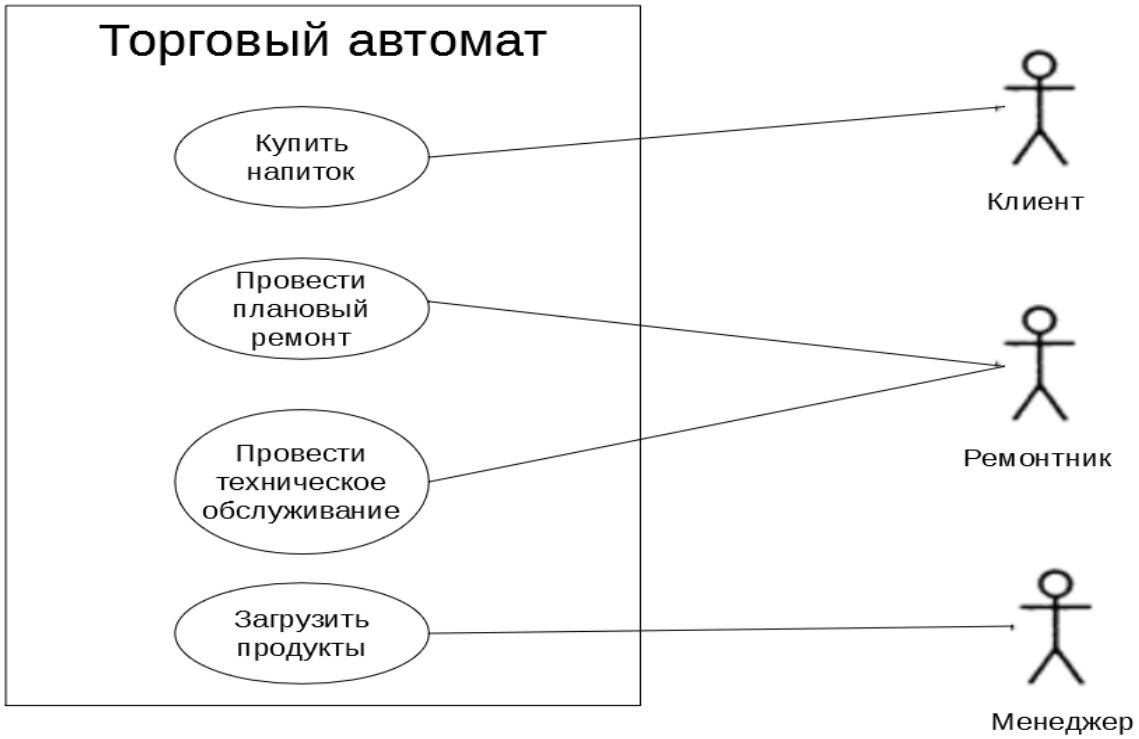


Рисунок 4.19 – Варианты использования для торгового автомата

### 4.5.2 Модели деятельности

Диаграмма деятельности показывает последовательность этапов, образующих сложный процесс, например, вычислительный алгоритм или технологический процесс. Диаграммы деятельности особенно полезны на ранних этапах проектирования алгоритмов и технологических процессов.

На рис. 4.20 показана диаграмма деятельности обработки заказа на покупку акций, принятого сетевой брокерской системой. Прямоугольники со скругленными углами — это виды деятельности, порядок которых обозначается стрелками.

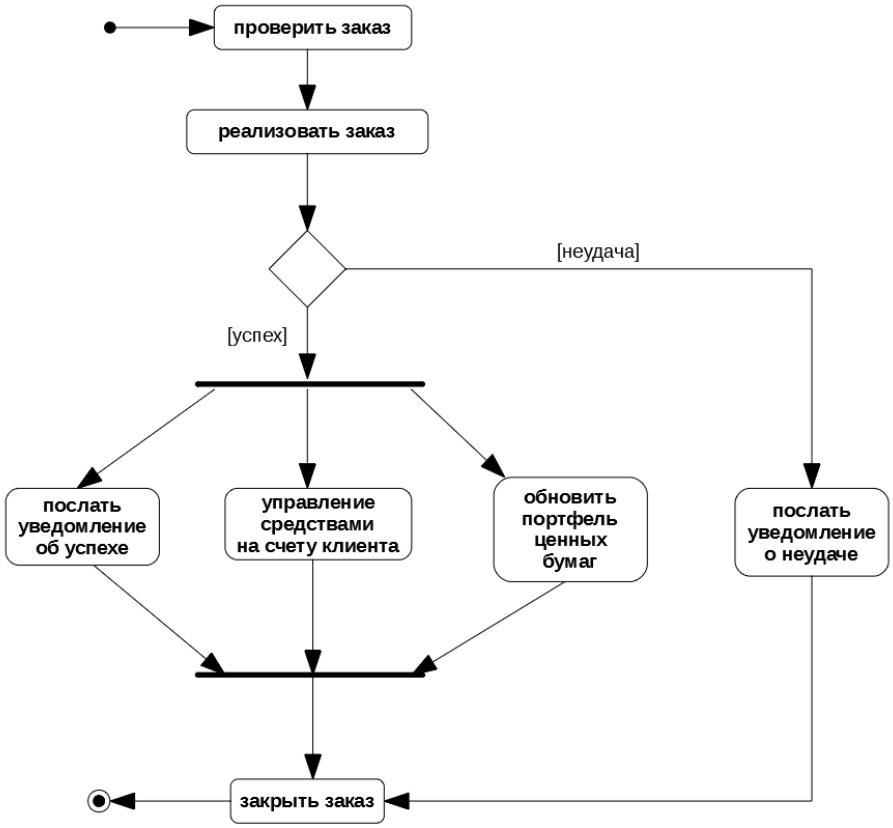


Рисунок 4.20 – Диаграмма деятельности брокерской системы при обработке заказа

Сетевая брокерская система начинает с проверки заказа. Затем заказ реализуется на фондовой бирже. Если выполнение заказа проходит успешно, система выполняет три действия одновременно: отправляет клиенту подтверждение, обновляет сетевой портфель ценных бумаг с учетом результатов сделки и заканчивает сделку со второй стороной, снимая средства со счета клиента и перечисляя наличные или ценные бумаги. Когда все три параллельных потока завершаются, система объединяет их в один поток и закрывает заказ. Если же выполнение заказа оказывается неудачным, система отправляет сообщение об этом клиенту и тоже закрывает заказ.

Диаграмма деятельности похожа на обычную блок-схему, потому что тоже показывает поток управления от этапа к этапу. Однако в отличие от блок-схемы, диаграмма деятельности может показывать одновременно параллельную и последовательную деятельность. Это отличие важно для распределенных систем. Диаграммы деятельности часто используются для моделирования организаций, потому что последние состоят из множества объектов (людей и организационных единиц), одновременно выполняющих множество операций.

Элементами диаграммы деятельности являются операции, а именно виды деятельности из модели состояний. Назначение диаграммы деятельности состоит в том, чтобы показать этапы сложного процесса и наложенные на них упорядочивающие ограничения.

Некоторые виды деятельности выполняются до тех пор, пока не будут прерваны каким-либо внешним событием, однако в большинстве случаев деятельность имеет собственное логическое завершение. Завершение деятельности является событием

завершения, которое обычно означает возможность начала выполнения следующей деятельности. Стрелка без надписи, соединяющая две деятельности на диаграмме, означает, что вторая деятельность может начаться только после того, как закончится первая.

Деятельность может быть разложена на более мелкие составляющие. Важно, чтобы виды деятельности на одной диаграмме относились к одному и тому же уровню детализации. Сохранить уровень детализации можно, если показывать составляющие высокоуровневой деятельности на отдельных диаграммах.

Если какая-либо деятельность имеет несколько последующих элементов, около каждой стрелки можно указать условие в квадратных скобках, например, «[отказ]». Условия проверяются после завершения предшествующей деятельности. Если одно условие оказывается выполненным, стрелка, около которой оно стоит, указывает на деятельность, подлежащую выполнению. Если не удовлетворено ни одно из условий, модель можно считать плохо согласованной, потому что система зависнет, если только не получит прерывание на более высоком уровне. Во избежание подобных ситуаций следует использовать условие «[иначе]». Если выполнено несколько условий, запущена будет только одна деятельность, причем невозможно определить, какая именно. В некоторых случаях подобная неопределенность закладывается осознанно, но часто она указывает на ошибку разработчика, который должен был учесть возможность перекрытия условий.

Для удобства в качестве обозначения множественного ветвления используется ромб с расходящимися от него стрелками, однако смысл его тот же, что и у стрелок, расходящихся от предшествующего состояния. На рис. 4.20 такой ромб имеет одну входящую стрелку и две исходящие, каждая из которых снабжена условием. При фактическом выполнении системы выбирается только одна из этих стрелок. Если несколько стрелок сходятся у некоторой деятельности, альтернативные пути выполнения сливаются. Здесь снова можно использовать ромб со сходящимися к нему стрелками.

Сплошной кружок с исходящей стрелкой обозначает начало выполнения диаграммы деятельности. Когда диаграмма деятельности активируется, управление передается на этот кружок, а затем переходит по исходящей из него стрелке к первым этапам деятельности. Завершение диаграммы обозначается символом «бычий глаз» (сплошной кружок с кольцом вокруг него). Этот символ не может иметь исходящих стрелок.

В отличие от традиционных блок-схем организации и компьютерные системы могут выполнять несколько видов деятельности одновременно. Скорость выполнения деятельности может меняться с течением времени. Например, за одной деятельностью может следовать другая (последовательное управление), после чего может выполниться несколько параллельных видов деятельности (разделение управления), а затем они снова объединятся в одну деятельность (слияние управления). Развилка и слияние обозначаются толстой линией с одной или несколькими входящими стрелками и одной или несколькими исходящими стрелками. Для синхронизации необходимо, чтобы управление было передано в точку слияния по всем стрелкам. На развилке управление передается всем параллельным видам деятельности одновременно.

## Контрольные вопросы к 4 главе

1. Что является фундаментальным элементом в объектно-ориентированном проектировании?
2. Что означает индивидуальность в объектно-ориентированном проектировании?
3. Что означает классификация в объектно-ориентированном проектировании?
4. Что означает наследование в объектно-ориентированном проектировании?
5. Что означает полиморфизм в объектно-ориентированном проектировании?
6. Из каких этапов состоит объектно-ориентированный подход?
7. Какой синтаксис сейчас используется для отображения объектно-ориентированных моделей проектирования?
8. Что означает концепция абстракции в объектно-ориентированном проектировании?
9. Что означает концепция инкапсуляции в объектно-ориентированном проектировании?
10. Что означает концепция объединения данных и поведения в объектно-ориентированном проектировании?
11. Что означает концепция совместного использования в объектно-ориентированном проектировании?
12. Что означает концепция выделения сущности объекта в объектно-ориентированном проектировании?
13. Что означает концепция «когда целое больше суммы частей» в объектно-ориентированном проектировании?
14. Какие типы моделей используются для полного описания систем в объектно-ориентированном проектировании?
15. Что такое событие?
16. Что такое событие сигнала?
17. Что такое событие изменения?
18. Что такое событие времени?
19. Как обозначается состояние в стандарте UML?
20. Что такое переход?
21. Какие условия называются сторожевыми?
22. Что изображается на диаграмме состояний?
23. Какая деятельность называется текущей?
24. Что такое объект и класс в объектно-ориентированном проектировании?
25. Как отображаются атрибуты и операции класса в нотации UML?
26. Что такое связь и ассоциация в объектно-ориентированном проектировании?
27. Что такое кратность связи?
28. Могут ли связи ассоциаций быть описаны атрибутами?
29. Какая ассоциация называется квалифицированной?
30. Что такое обобщение?
31. Какие цели преследует обобщение?
32. Кто может называться действующим лицом?
33. Как называются различные взаимодействия действующих лиц с системой?
34. Являются ли сбойные ситуации частью варианта использования?
35. Что показывает диаграмма деятельности?
36. Можно ли использовать диаграмму деятельности для параллельных алгоритмов действий?

## Заключение

Активное развитие информационных технологий создает предпосылки для создания и использования информационных систем на разных предприятиях. Несмотря на большое количество представленных на российском рынке ИС, руководителям предприятий приходится принимать решение о разработке собственной ИС. Основная причина такого шага заключается в том, что автоматизируются только те функции, которые необходимы в рамках конкретной деятельности фирмы.

Если на предприятии для создания ИС выделяется небольшой ограниченный бюджет и ставятся жесткие сроки, то разработчики применяют структурный подход к проектированию ИС. В этом случае основной упор идет на функциональную составляющую системы, что приводит к необходимости постоянной поддержки работоспособности ИС в актуальном состоянии.

Когда на предприятии заинтересованы в создании стабильного программного продукта, то разработчики используют объектно-ориентированный подход к проектированию ИС. В проект ИС закладывается модель данных всей предметной области, в которой функционирует предприятие и набор операций с данными, применимый практически во всех мыслимых возможных ситуациях.

В обоих случаях ИС разрабатывается в коллективе специалистов-разработчиков, что подразумевает применение единых стандартов выходных спецификаций проекта, программного кода и оформления документации для совместной работы.

Умение синтаксически и методически правильно применять стандартные нотации для создания проектов ИС и является основной целью данной дисциплины. Совместное использование полученных знаний в рамках данной дисциплины и дисциплин по системному анализу, программированию и баз данных дает студенту возможность принимать непосредственное участие в разработке любых информационных систем.

## Глоссарий

***IDEF0-методология*** – совокупность методов, правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области.

***UML*** (Unified Modeling Language) – стандарт обозначений для объектно-ориентированного моделирования.

***Ассоциация*** – это описание группы связей, обладающих общей структурой и общей семантикой.

***Атрибут*** – это именованное свойство класса, описывающее значение, которое может иметь каждый объект класса.

***Базовый элемент системы*** – элемент, описания которого дальнейшему делению не подлежат, то есть их описание уже известно.

***Вариант использования*** – это связный элемент функциональности, представляемый системой при взаимодействии с действующими лицами.

***Восходящее проектирование*** – вид проектирования, когда решение задач низких иерархических уровней предшествует решению задач более высоких иерархических уровней.

***Действие*** – это ссылка на поведение, выполняемое в ответ на произошедшее событие.

***Действующее лицо*** – это непосредственный внешний пользователь системы.

***Деятельность*** – это фактическое поведение, которое может вызываться любым количеством действий.

***Диаграмма классов*** – это граф, вершинами которого являются классы, а ребрами – их отношения.

***Диаграмма состояний*** – это граф, узлами которого являются состояния, а направленными дугами – переходы между состояниями.

***Доминирование*** – относительный порядок блоков на IDEF0-диаграмме с размещением их по степени важности, как ее понимает автор диаграммы.

***Индивидуальность*** – деление данных на дискретные сущности-объекты, хорошо отличимые друг от друга.

***Информационная система*** – это организационно-техническая система, использующая информационные технологии в целях обучения, информационно-аналитического обеспечения человеческой деятельности и процессов управления.

***Итерационное проектирование*** – вид проектирования, когда требуется повторное выполнение проектных процедур предыдущих этапов после выполнения проектных процедур последующих этапов.

***Квалифицированная ассоциация*** – ассоциация, у которой имеется специальный атрибут-квалификатор, используемый для того, чтобы отличать друг от друга объекты, находящиеся на полюсе ассоциации с кратностью «много».

***Класс*** – описание группы объектов с одинаковыми свойствами, поведением, типами отношений и семантикой.

***Класс ассоциаций*** – это ассоциация, которая одновременно является классом.

***Классификация*** – процесс группировки объектов с одинаковыми структурами данных и поведением в классы.

***Количество информации*** – это мера уменьшения энтропии объекта после совершения некоторого события.

***Компонента системы*** – часть системы, не обладающая её свойствами, а представляющая собой обычную совокупность однородных элементов.

***Кратность*** – это количество экземпляров одного класса, которые могут быть связаны с одним экземпляром другого класса через одну ассоциацию.

***Математическая модель технического объекта*** – система математических объектов (чисел, переменных, матриц, множеств и т.п.) и отношений между ними, отражающих некоторые свойства технического объекта.

***Наследование*** – это наличие у разных классов, образующих иерархию, общих атрибутов и операций.

***Нисходящее проектирование*** – вид проектирования, когда решение задач высоких иерархических уровней предшествует решению задач более низких иерархических уровней.

***Обобщение*** — это отношение между классом (суперклассом) и одной или несколькими его вариациями (подклассами).

***Объект*** – это концепция, абстракция или сущность, обладающая индивидуальностью и имеющая смысл в рамках системы.

***Один бит*** – это количество информации, получаемое при осуществлении одного из двух равновероятных событий.

***Операция*** – это функция или процедура, которая может быть применена к объектам класса.

***Параметр модели системы*** – количественное выражение свойств системы в модели.

***Переход*** – это мгновенная смена одного состояния другим.

***Подсистема*** – это относительно независимая часть системы, которая обладает всеми свойствами системы и, в частности, имеет свою подцель, на достижение которой эта подсистема и ориентирована.

***Полиморфизм*** – особенность у операций, подразумевающая разное поведение в разных классах.

***Принцип декомпозиции*** – разбиение представлений каждого уровня на ряд составных частей-блоков с возможностью раздельного проектирования этого уровня.

***Принцип иерархичности*** – структурирование представлений о системе проектирования по степени детальности описаний.

***Проект информационной системы*** – проектно-конструкторская и технологическая документация, в которой представлено описание проектных решений по созданию и эксплуатации ИС в конкретной программно-технической среде.

***Проектирование информационной системы*** – процесс преобразования входной информации об объекте проектирования, о методах проектирования и об опыте проектирования объектов аналогичного назначения в соответствии со стандартами в проект информационной системы.

***Родительская диаграмма*** – это более общая диаграмма на некотором шаге декомпозиции относительно более детальной диаграммы.

***Свойство объекта*** – сторона объекта, обуславливающая его отличие от других объектов или сходство с ними и управляющуюся при взаимодействии с другими объектами.

***Связь*** – это физическое или концептуальное соединение между объектами.

***Сигнал*** (в информационном смысле) – материальный носитель информации, то есть средство переноса информации в пространстве и во времени.

***Сигнал*** (в нотации UML) – это явная односторонняя передача информации от одного объекта к другому.

***Система*** – это конечное множество функциональных элементов и отношений между ними, выделяемое из среды в соответствии с определенной целью в рамках определенного временного интервала.

***Событие*** – это происшествие, случившееся в определенный момент времени.

***Событие времени*** – это событие, вызванное достижением момента абсолютного времени или истечением временного интервала.

***Событие изменения*** – это событие, вызванное выполнением логического выражения.

***Событие сигнала*** – это событие получения или отправки сигнала.

***Сообщение*** – совокупность знаков или периодических сигналов, содержащих информацию.

***Состояние*** – это абстракция значений и связей объекта.

***Сторожевое условие*** – это логическое выражение, которое должно быть истинным, чтобы переход мог запуститься.

***Текущая деятельность*** – деятельность, занимающая некоторый промежуток времени.

***Технология проектирования*** – это совокупность методологии и средств проектирования информационной системы, а также методов и средств организации проектирования.

***Типовая проектная процедура*** – процедура, предназначенная для многократного применения при проектировании многих типов систем.

***Унифицированный элемент*** – стандартизированный элемент системы, ориентированный на применение в ряде различных систем определенного класса.

***Управление объектом*** – процесс формирования целенаправленного поведения системы посредством информационных воздействий, вырабатываемых человеком или устройством.

***Элемент системы*** – некоторый объект, обладающий рядом важных свойств и реализующий в системе определенный закон функционирования, причем, внутренняя структура данного объекта не рассматривается.

***Энтропия*** – это мера неопределенности какого-либо опыта, который может иметь разные исходы.

## Список литературы

1. Кориков А.М. Теория систем и системный анализ: учебное пособие для вузов / А.М. Кориков, С.Н. Павлов. – Томск: ТУСУР, 2007. – 343 с.

2. Павлов С.Н. Теория систем и системный анализ: Учебное пособие / С.Н. Павлов. – Томск: Томский межвузовский центр дистанционного образования, 2003. – 134 с.

3. Смирнова Г.Н. Проектирование экономических информационных систем: Учебник. / Г.Н. Смирнова, А.А. Сорокин, Ю.Ф. Тельнов. – М.: Финансы и статистика, 2002. – 512 с.

4. Описание стандартов семейства IDEF [Электронный ресурс]. – URL: http://idef.ru/ (дата обращения: 10 ноября 2023). – Режим доступа: свободный.

5. Рамбо Д. UML 2.0. Объектно-ориентированное моделирование и разработка. 2-е изд. / Д. Рамбо, М. Блаха. – СПб: Питер, 2007. – 544 с.

6. Object Management Group [Электронный ресурс]. – URL: http://omg.org (дата обращения: 10 ноября 2023). – Режим доступа: свободный.